UNIVERSITY OF CALGARY

A Systematic Literature Review of Software Engineering for Scientific and Engineering Software and an Industrial Oil Pipeline Software Case Study

by

Roshanak Farhoodi

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

AUGUST, 2011

Your file Votre référence
ISBN: 978-0-499-00287-7

Our file Notre référence
ISBN: 978-0-499-00287-7

Abstract

Scientific and Engineering Software (SES) is different from commercial software, it often targets computational analysis of a problem without a prior solution. This work is aimed at systematically reviewing the literature for extracting particular challenges and solutions of SES development. We also conducted a case study, by developing decision support and optimization software for the oil industry to bring our findings of the review into practice and provide evidence of challenges/solutions of developing SES.

Our development experience confirmed observations of the literature on these challenges, especially those of requirement elicitation and testing. The requirements in SES are often unknown upfront, developers are often domain experts and software validation is complex; for the scientific/engineering core for which no certain test oracle exists.

Following software engineering practices, such as adopting object-oriented technology, iterative development approach, MVC architectural pattern, unit and GUI testing, we could successfully develop and commercialize the optimization software system.

Acknowledgements

My deepest gratefulness goes to Dr. Vahid Garousi for his intense support, encouragement and guidance throughout this work. Accomplishing this work would have been impossible without his supervision and assistance.

I would also like to thank Dr. Michael Smith, Dr. Diwakar Krishnamurthy, and Dr. Jonathan Paul Sillito, my thesis committee members, for suggesting revisions to this work and providing useful help and feedbacks.

I would like to express my heartfelt sincere gratitude and appreciation to my lovely parents Mohammad and Shahla, my wonderful siblings Alireza, Ramin and Rozita and my dear fiancé Masoud for their unstoppable support, love and encouragement.

I would like to thank all my dear friends in Calgary, especially SoftQual research group members; Shahnewaz, Christian and Ehsan, who closely brought me joy and support during those tough days.

This work was financially supported through the Alberta Ingenuity New Faculty Award no. 200600673, which is greatly acknowledged.

To Tranquility and Peace

Table of Contents

List of Tables

List of Figures

List of Acronyms

CLR	Common Language Runtime
EBSE	Evidence-Based Software Engineering
GUI	Graphical User Interface
IL	Intermediate Language
KML	Keyhole Markup Language
LOC	Lines Of Code
MVC	Model-View-Controller
SCADA	Supervisory Control And Data Acquisition
SDLC	Software Development Life Cycle
SE	Software Engineering
SES	Scientific and Engineering Software
SLR	Systematic Literature Review
SS	Scientific Software
SUT	System Under Test

Chapter One: **Introduction**

In this chapter, we introduce the problem and the research gaps we intend to address throughout this thesis. We discuss the motivation and goals, followed by presenting the contributions of the thesis. The thesis organization is provided at the end of this chapter.

1.1 **Introduction and Motivation**

Software systems are one of, if not, the most critical parts of any modern system (e.g., scientific, engineering, health-care, and military). It is hard to think of large-scale industrial control and monitoring systems, manufacturing plants, rocket and airplane navigation systems and many more medical, chemical, electrical and mechanical systems without a software backbone.

In this thesis we focus on Scientific and Engineering Software (SES) in particular. We investigate different publications on the methods that the scientists, engineers, or professional developers use as well as the issues, challenges, experiences and insights they reported during the life cycle of SES.

SES systems may exhibit quality and functionality shortfalls and failures besides timescale and effort overruns [1, 2] as they are not usually developed by professional software engineers [3-5]. Scientist and (non-software) engineers usually face challenges while interacting with software engineers in building SES systems [6], which results in a gap between these two communities, the so called "software chasm" by Kelly [3]. Also, as stated by Cremer et al. [7]: "*while advances in hardware for scientific computation continue to be made, the process of creating scientific software that takes full advantage of the hardware remains a time-consuming, error prone and expensive art*". In this study, we aim at gathering, reviewing and aggregating practical and theoretical evidence as presented by different authors to identify the root causes of the current gap and problematic issues and also to extract how they can be possibly addressed.

A software (a computer program) is considered scientific software if the subject it addresses is scientific, e.g., mathematical programs such as discrete Fourier transformation calculator. Similarly, a software or program is considered engineering software if the subject it addresses is related to engineering, e.g., power plant control software. Of course, the boundary between these two categories can be often very slim or inseparable. Segal believes "*the major difference between scientific software and other commercial software lies in the complexity of the domain*" [8].

Scientists and engineers have been developing software for their own specific use for over six decades now. Starting from early 1950, developers at the US Department of Defence (DoD) developed scientific software for the analysis of defence systems [9] to the recent huge software systems built to better study and analyze climate change [10]. Numerous companies focus on this very critical business of developing SES covering all different engineering disciplines, e.g., Fekete Inc. which develops oil reservoir analysis software tools [11], Energy Solutions International developing oil and gas software [12], Engineering Software Center developing various engineering applications [13], and Intuitive Software developing structural engineering software [14]. In both research and industrial communities, software engineering methodologies and techniques are being adapted more and more into the development of major systems in areas such as aero space, medical and embedded systems [15-17].

Scientific workshops such as the Workshop on Software Research and Climate Change [18], Software Engineering for Automotive Systems [19], Workshop on Software Engineering in Health Care [15], Workshop on Aerospace Software Engineering [16], and Workshop on Software Engineering for Computational Science and Engineering [17] are being held frequently in this field. The recent findings, techniques and also challenges in developing SES are being discussed there and also to incorporate the latest finding of the software engineering community in those application domains.

There are frequent stories about failure of software systems, e.g., Toyota's break system failure [2], Mars Climate Orbiter crash in 1999 [1], death resulted from inadequate testing of the London Ambulance Service software [1] and China Airlines Airbus Industries A300 crash in 1994 [1]. Increasing challenges of building defect-free software is one of the main reasons of bringing software engineering best practices into developing SES [3]. Yet there still exists a gap between how scientists/engineers and software engineers look at the issues of developing SES [20].

In the first part of the current work, we present the results of a systematic mapping study followed by a systematic literature review (SLR) [21, 22] conducted in the area of SES. We have undertaken this systematic literature review to identify the strengths and weaknesses of the state of the art and practice in developing SES besides highlighting the challenges of past, current and future trends from the perspective of developers, researchers and scientists. This is achieved by extracting and aggregating evidence from key publications in this field and summarizing their insights and findings towards improving the quality and efficiency of scientific software engineering tasks. We also identified the best practices reported which are applicable to different software development phases in various problem domains.

By conducting the SLR we were able to characterize SES as a type of software having four main differences from commercial software. First, in SES development the requirements cannot be decided in advance, because in most cases the objective of developing the software is to find the solution to a problem for which no prior solution exists [23, 24]. Second, as the main objective of developing SES is to provide a correct and reliable code which can be utilized to improve the target science or engineering discipline, the factor of building a working system in the shortest amount of time often outweighs adopting rigorous software engineering practices to ensure the quality of end product [23, 24]. Third, the developers of SES are mostly domain experts (i.e. scientists and engineers) rather than professional software developers [23, 24]. Finally, testing SES has two independent stages; testing the scientific/engineering core for which usually no certain test oracle exists, and testing the software that provides access to that scientific/engineering core. These four distinctive characteristics introduce unique challenges to the development of SES, which require particular considerations to be addressed.

In the second part of the thesis, we brought the insights taken from systematic literature review into practice by conducting an industrial case study. This case study was a part of a bigger project to develop industrial engineering software for the optimization of oil pipeline operation. In the case study, we planned to practically experience the challenges of SES development, to utilise the solutions reported, to verify the applicability of the best practices and to investigate their adaptability, where relevant. As a result, we presented and discussed the experience of developing the engineering decision support and optimization software and when relevant, relate and compare our experience with that of the literature as well as reporting the specific challenges faced. In this case study, phases of software development are demonstrated and discussed, mainly with the aim of providing evidence on the challenges of developing engineering software, verifying the applicability of the best practices found in the literature, investigating their adaptability and validating the solutions reported.

1.2 **Contributions of the Thesis**

To the best of our knowledge, there is no other work on the aggregation of the literature on the state of the art and practice of software engineering for SES. The importance of systematic mapping studies and systematic literature reviews (will be discussed in Chapter 2) as well as the importance of software engineering for SES besides the lack of a comprehensive review in the field inspired us to aggregate well-known resources into one work. To achieve this goal, which is half of the contributions of this thesis, we identified a group of important research questions and followed the precise guideline of performing systematic reviews. We aggregated the challenges, solutions and observations reported in the literature for the different phases of software development and extracted the best practices provided for improving SES development. The findings are provided, along with their particular context and domain, to give the reader a precise understanding about their applicability and generalizability in various situations. In this way, these findings can serve as a reference for other researchers and practitioners who are interested in SES design and development. By reviewing the challenges identified, we were able to characterize SES as a certain type of software,which deals with the problems in scientific and engineering context that were frequently mentioned as being in a complex domain for the typical professional software engineer to learn and master. Similar to other Enterprise software, it is expected that the designing and implementing SES can be improved by using standard software engineering practices, such as adopting OO methodology and design and architectural patterns. However, testing SES is yet an open issue, mainly as the result of not having easy access to test oracles associated with validating the scientific core.

For the practical part of the thesis, studying the development of oil pipeline optimization and decision support software, we adopted the best practices we identified in the literature whenever applicable to different development phases of our system. We studied and summarized the real world challenges of developing engineering software. None of the studies presented in the SLR section of this work were preceded by a comprehensive literature review, to benefit the experiences and evidence presented by other researchers in this context.

The software requirements for this system were analysed and designed, the software features and functionality were implemented and tested in an iterative and flexible manner to make the practice flexible and maintainable enough with regard to the characteristics of SES. This case study was also conducted in order to provide evidence on SES development, in addition to what we found in the literature. We experienced the complexity of the requirement elicitation and decided to adopt an iterative development approach to address the emerging requirements received from the domain expert. We had the opportunity to verify our understanding of the domain and the problem as we moved toward the completion of the development.

The developed application for oil pipeline operation optimization is being commercialized currently. The demo of the application has been presented to a group of potential customers and they have shown initial interest for the customization and utilization of the system in their company.

1.3 **Thesis Organization**

The remainder of this thesis is organized as follows. The background information and related work is provided in Chapter 2. In Chapter 3, the research method used to perform the review is presented, the research questions that the review tried to answer detailed and the results extracted from the pool of the primary studies are provided. In Chapter 4, the overview of the case study for developing pipeline operation optimization software is presented as the real world practice of developing SES. The details related to analysis and design, development and testing of the software application are given in Chapter 5, 6 and 7 respectively. Chapter 8 discusses the operation and usage scenarios of the system. Finally, Chapter 9 concludes the whole thesis and presents the future work directions.

Chapter Two: **Background and Related Work**

In this section, we provide brief backgrounds on software engineering for SES (Section 2.1) and on systematic literature reviews (Section 2.2). We then summarize the related works in Section 2.3.

2.1 **Developing SES**

Tang quoted the definition of SES from Smith [25] in her thesis as "the use of computer tools to analyze or simulate continuous mathematical models of real-world systems of engineering or scientific importance so that we can better understand and predict the system's behaviour" [26]. SES's have a large variety: They are either expensive commercial software packages which address a vast diversity of problems and domains (e.g., Matlab, and Maple) and may have a large dedicated software engineering team behind them. There are also non-commercial SES tools which are free and/or open-source such as the R project for statistical computing [27], or Gretl (GNU Regression, Econometrics and Time-series Library) [28].

More often, scientists attempt to develop scientific software to provide themselves or their colleagues a tool to better analyze and understand what's happening in a certain scientific phenomenon (e.g., [29-32]).

In some cases, the entire purpose of developing the software is to solve a problem that does not currently have a solution [23]. In such cases, the validation of the end product is more complex than the case where the goal is to better understand a phenomenon, because predicting the expected results to use as a test oracle is very hard at times (if not impossible). The role of the expert's observations and theoretical basics in those cases becomes significant.

A general process model of scientific software development is shown in Figure 1. A vague idea as to the solution to a scientific or engineering problem or the simulation of a real physical system usually triggers the specification of some primary requirement which is the starting point of developing the primary version of the software.

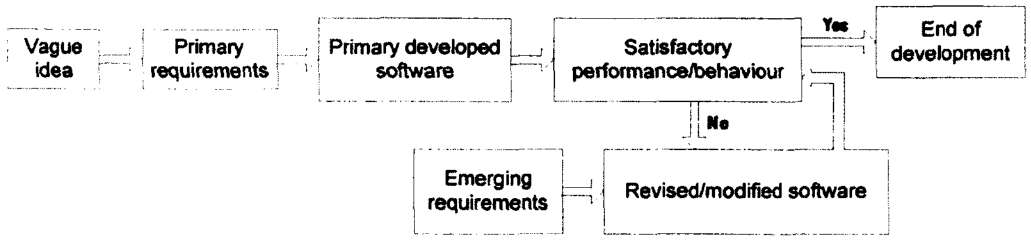

Vague idea
Primary requirements
Primary developed software
Satisfactory performance/behaviour
Yes
End of development
No
Emerging requirements
Revised/modified software

Figure 1: A general process model for scientific software development (inspired by ideas from [33])

Once a prototype version of the software is developed and is executed, based on certain criteria (as defined in the system's functional and non-functional requirement specifications), the output(s), performance and behaviour of the software are evaluated. If it is found unsatisfactory, the software is reworked and modified according to the further elicitation of requirements which will normally result in a more precise design and implementation of the software.

This revision cycle will be repeated until a satisfactory and rational behaviour is obtained which will represent the end of the development process. This resembles the well-known iterative software development process except that, in the process of developing SES, most of the time the requirements are emerging as the system is being developed with the developer may not have a sound understanding about them at the beginning.

Typically scientists who develop software do not have much knowledge and/or interest in the software engineering aspects of their product [34]. Instead, as stated by Chilana [35], they are willing to investigate the use of computational tools to help or improve the understanding of a scientific concept. On the other hand, the result of the survey by Maxville [36] confirms that researchers and scientists are "open to effective techniques that can improve communication, transparency and quality".

Table 1 summarizes some important differences between SES and typical conventional software. In the next sections, our SLR results provide more details on each of these characteristics.

	Developer Background	**Domain Dependency**	**Requirement Specification**	**Testing**	**Software Users**	**Maintenance**
Conventional Software	Software engineering	Mostly Independent	(Usually) Comprehensively defined	Systematic	(Usually) Public	Systematic
SES	Scientific and engineering disciplines	Very dependent	Loosely defined	Ad hoc	Scientists and engineers	Ad hoc

Table 1: Comparing the characteristics of conventional software vs. SES

To clarify the usage of the terms "engineer" and "engineering" in the rest of this thesis, when they are used alone, we denote the conventional engineers (e.g., mechanical, electrical, chemical, etc.) and the general area of engineering. However, when software engineer or software engineering is used, the software focus areas are meant.

2.2 Systematic Mapping Studies and Systematic Literature Reviews

Reviews of research literature are carried on for a variety of goals. Usually the reviews are conducted to provide the theoretical background for subsequent research activities or to learn the breadth and depth of the research in a certain topic or context. The reviews also aim at answering practical research questions by summarizing and presenting what existing research has to offer [37]. As a result, we often find the research reviews in the introduction section of the publications.

However, there exists another type of literature review that is considered an original and important type of research by itself [38]. Rather than providing a base for the researcher to be able to conduct further research and investigations, it builds a solid starting point for other researchers and practitioners interested in a particular subject. Four fundamental characteristics are identified for defining these types of reviews, which make them different from other conventional reviews [38]. They are (1) systematic, which means they have to follow a methodological approach, (2) explicit, which means the procedure by which the review was conducted should be explained clearly, (3) comprehensive, which means the review is expected to include all existing relevant material, and as a result (4) reproducible by others.

The definition of SLR is summarized by Fink in his book [38] as "a systematic, explicit, comprehensive, and reproducible method for identifying, evaluating, and synthesizing the existing body of completed and recorded work produced by researchers, scholars, and practitioners".

Kitchenham et al. in [39] have discussed the importance and educational and scientific values of evidence-based software engineering (EBSE). SLRs in their paper are considered one of the primary tools of EBSE and systematic mapping is mentioned to be a certain type of SLR which is used as a good starting point for more detailed studies. Evidence-based software engineering, primarily inspired from evidence-based clinical medicine, aims at employing practical evidence as a guide to the adoption of software development procedures and practices [40].

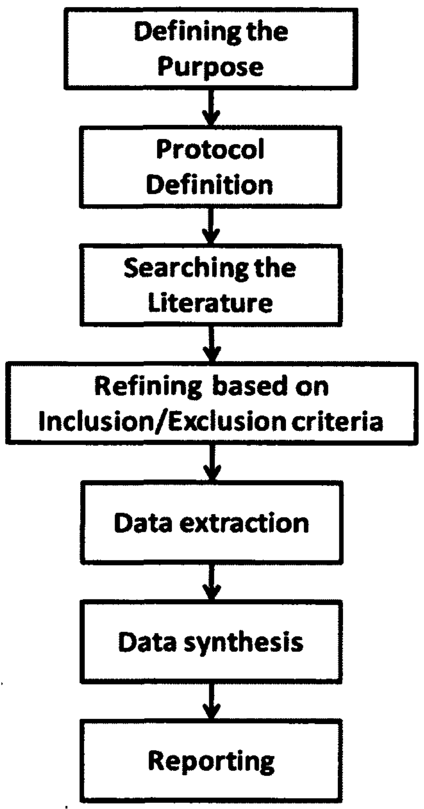


Figure 2: Steps of conducting SLR

In order to properly conduct a SLR a set of particular steps needs to be taken. These steps are demonstrated in Figure 2. The first step is to clearly define the purpose and motivation of conducting the review. As mentioned earlier, a SLR is a method for investigating existing publications based on a defined protocol unlike surveys or other conventional literature reviews that aim at briefly presenting the current advances and research results,. Hence, the methodical approach for conducting the SLR is characterized by that protocol, and should be defined in the beginning of the review, to state why and how the review is conducted [21].

The protocol is defined by asking a number of overarching research questions and the rest of the review focuses on investigating how well or to what extent each of the selected publications answers the questions of interest. Meaningful criteria for including as many relevant publications as possible should be defined, to make the review comprehensive and complete.

The next step is searching the literature to find all potential publications from well-known publishers. In this step, a group of search keywords need to be defined, by which the relevant publications for the review can be found. After finding all the publications according to the search keywords, based on a defined inclusion/exclusion criterium, the decision of keeping or removing each of the papers from the pool of publications has to be made. Selected papers should then be read and investigated and their presented evidence related to each research question of the review should be extracted and summarized. In order to increase the reliability of the information extracted in this step and to avoid biased judgements and understandings, the work is required to be peer-reviewed and the personal uncertainties about the inclusion or relevance of each papers need to be discussed among the authors. The data is then aggregated using an appropriate method, such as narrative, meta-ethnography and thematic [41]. The aggregation result is then presented in different forms such as comparative charts, tables, figures along with related discussions. Finally the review will be concluded by stating the major findings and suggesting the possibility of further investigation and research.

SLRs are popular and appear in different areas of science and engineering, e.g., medical sciences (e.g., [42]), social sciences (e.g., [43]), mechanical engineering (e.g., [44]), and software engineering (see the survey in [22]). In designing and executing our systematic review, we have benefited from previous systematic reviews, especially the three recent ones published by Ali et al. [45], Harman et al. [46] and Engström et al. [47].

A software engineering mapping study (or systematic map) is a method to build a classification scheme for the software engineering field of interest [48]. It provides a structure of the type of research reports and results by categorizing and classifying them. A visual summary of the results will be generated at the end. The analysis of results focuses on frequencies of publications for categories within the scheme. Therefore, the coverage of the research field by existing literature can be determined. The main goal of the systematic mapping as stated by Peterson et al. [48] is to generate an overview of a certain research area and to identify the quantity and type of the available research and results. The study starts with defining research questions of interest, followed by gathering the relevant publications. The required data are then extracted and the results are presented as the outcome of the systematic map. The procedure for conducting a systematic map is demonstrated in Figure 3. As shown in the figure, first the research questions of interest are defined and then similar to conducting SLRs, the literature is searched for finding the relevant publications, using defined search keywords. The resulting pool of publications is then refined using the inclusion and exclusion criteria. The required data is extracted, classified and reported at the end.

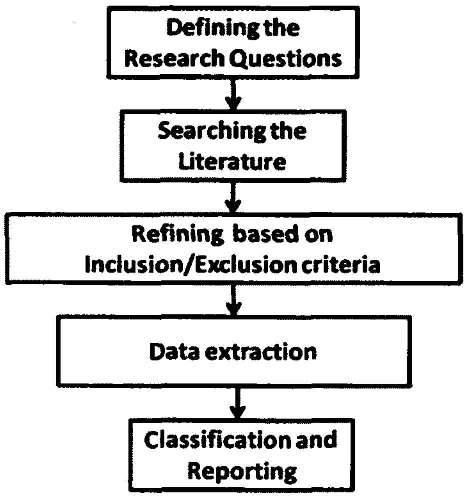

Defining the Research Questions
Searching the Literature
Refining based on Inclusion/Exclusion criteria
Data extraction
Classification and Reporting

Figure 3: Steps of conducting mapping study

The detailed steps of applying mentioned procedures for conducting SLR and mapping study is presented in Chapter 3.

2.3 Related Works

In this section, first we briefly present the available literature which either study or review SES development and then in the second part, we present several known commercial oil pipeline operation software applications.

2.3.1 *SLRs on SES Systems*

To the best of our knowledge, there is no systematic literature review on the software engineering for SES development. As a related work, we were only able to find a systematic mapping study by Feitosa et al. [49] on software engineering for embedded systems and mobile robot software development. They found out that the application of software engineering in this field is increasing over the years, though some areas such as software testing, reference architectures and aspect-oriented development still need more attention.

Among the papers, we found two other publications ([26, 50]) with the aim of extracting the state of the art in the field of SES. We considered them as "related work" in this section, as we aimed to separate the publications which comprehensively conducted research on SES development from the papers which investigated just a certain stage in SDLC or focused on one issue in the development process.

The first one was a thesis by Jin Tang [26] from McMaster University (Canada) in 2009, which can be considered as a comprehensive survey conducted in order to find out what SE methodologies and technologies are currently being used in SES. The author also aimed to identify the areas of improvement in the field and investigate whether the SES community is interested in adopting new "software engineering" ideas. Other information such as educational background, working experience, group size, software size, development practices and software quality were also gathered and reported in this work. 47% of the respondents to this survey were academic developers and 45% were developers from industry.

The second was also an online survey conducted by Hanney et al. [50] in 2008 which had around 2,000 respondents. The aim of their survey was to investigate how scientific software is being developed and used by the majority of the scientists. They also gathered the information on how scientists gain their software development knowledge and skills, the impact of team size on their development activities, the importance of testing and use of super-computers versus desktops and intermediate computers for using and developing their software. 50% of the respondents to this survey were academic developers and the rest have different occupations such as managers, supervisors, industrial research scientists, system administrators, laboratory technicians and clinicians.

Another related work is a technical report by Greenough et al. [51] which reviewed the practices in computational science and engineering department and also proposed a set of standards and best practices. They identified the elements of software development process, classified the software projects and introduced tools, techniques, methods and metrics to assure the quality of the end result.

In the subsequent sections (when relevant), we will provide more information reported in the related work to give the reader a broader view in the field of SES.

2.3.2 *Pipeline Operation Software*

In this section we briefly introduce a number of popular software tools, which are designed for addressing the needs of oil industry and are being widely used to better manage the pipeline operation by proposing facilities such as scheduling of oil products and refinery operations, and product distribution planning.

The features and brief description of the functionalities provided by these tools are summarized in Table 2. Among these tools, as mentioned in the table, EnergyOne is the most similar tool to our developed application, in the way that it also provides the users with the energy management solutions. Other tools are often being used for scheduling and oil plant management activities. All of the tools presented in the table are commercial large-scale tools.

Software name	Ref	Description
SCHEDULE++ PIPELINE & PORT	[52]	Scheduling of crude oil delivery and unloading at a port, multi-component batch composition and pipeline batch transportation scheduling for oil refineries that receive their raw components via shared multi-purpose pipelines from multiple ports or companies that are operating those pipelines
SIMTO Scheduling	[53]	Scheduling feedstock upload, storage and tank transfer and scheduling the feeds to process units, scheduling process units including production quality and reaction processes, scheduling blending and shipments of final products
EnergyOne	[54]	Customizable pipeline energy management system with integrated pipeline scheduling, can be customized to run independently or to work with other internally developed or commercial software pipeline management package
H/SCHED	[55]	Scheduling crude supply, refinery operations, product blending, and refinery product distribution. Any one or any combination of these features can be provided
Pipeline transporter	[56]	Used in integration with Primavera project management application and provide business process solutions that synchronizes project system, plant maintenance and production planning modules

Table 2: Popular oil software solutions

2.4 Chapter Summary

In this chapter, the general knowledge about SES development was offered. The systematic mapping studies and systematic literature reviews were introduced and their educational and scientific values were mentioned. SLR is mentioned to be one of the very common forms of EBSE, providing valuable information for scientists and practitioners, which is a methodical, comprehensive and organized review about the state of the art in a particular domain and about a certain subject. Systematic mappings are also a good starting point for more detailed studies as they categorize different types of primary studies and give summary of the results.

Related works in the field of software engineering for SES were presented in the second part of this chapter, followed by a brief introduction of the well-known commercial oil pipeline operation software packages.

In the next chapter we will elaborate the details of conducting systematic mapping and SLR on the development of SES and presents and discuss the findings and results.

Chapter Three: **Systematic Mapping Study and SLR**

In this chapter, we present the process of performing systematic mapping and systematic literature review on software engineering for SES development besides the results. The goal, research questions and results extracted from the literature in this study are discussed in Section 3.2. Our selection strategy to choose articles is presented in Section 3.3. The data extraction approach is presented in Section 3.4. In Section 3.5 we discuss the choice of our aggregation method and in Section 3.6 we present the results found. Section 3.7 presents briefly a discussion on the threads to the validity of this review.

3.1 **Research Method**

We have performed a systematic mapping study, later extended to a SLR, for assessing and investigating the state of the art and practice in software engineering for SES development. This SLR is carried out using methods inspired from the guidelines provided by Kitchenham and Charters [21] as explained in Chapter 2.

3.2 **Goal and Research Questions**

The goal of our study is to review the state of the art in SES, identify the weaknesses and strengths, highlight the challenges and find out the future trends and directions in this field from the point of view of SES developers, researchers and scientists.

Based on the above goal, we raised the following research questions. To extract more detailed information for each of the questions, each question is broken down into several sub-questions.

RQ 1- What are the demographics of studies (research space) in SES?

This RQ aims at gathering demographic information from the papers under study, in order to provide the reader with various classifications related to authors, goals of the papers, trends of publications and affiliations of the research groups. Information such as application domains, size and complexity of the applications and number of the publications of each author are extracted. This RQ is the systematic mapping component of the study. Performing these types of demographic analysis has been popular in empirical software engineering studies and is being frequently conducted by other researchers [46, 57-59]. The sub-questions are:

RQ 1.1- What is the trend of the publications?

The trend of the publications in the field shows how the research activities within the area have been changing in a particular period. As a result, the increasing trend can show this field is gaining more attention in recent years, and the body of knowledge is growing in this area.

RQ 1.2- What are the application domains? (e.g., command and control, chemical engineering, mathematics)

SES development is an interdisciplinary research area and by extracting the disciplines which are frequently demanding the development of software systems, more rigorous domain-dependent practices and frameworks can be suggested to better fit this demand.

RQ 1.3- What is the size of software systems under study?

By extracting the size of the software systems in the publications, a general overview on the scale of SES system in the publications can be achieved. This can motivate the development of specific practices to fit SES development based on the context they are expected to be used, e.g. large-scale software systems need different considerations in their development process compared to mid/small-scale systems.

RQ 1.4- Which programming languages are reported in each paper?

Identification of the programming languages which are being used frequently, besides giving an overview of the popular languages in this field, can define new research directions to address certain issues of implementation using these languages and deal with interoperability concerns in this context.

RQ 1.5- Which authors and countries have been more active

The ranking of the scholars based on the number of their publication in the field of SES development can potentially be used by researchers and grad students [57]. Knowing the name of the active authors who frequently publish in the field, grad students and researchers can easily find and investigate their publications and establish further collaboration with the authors. Knowing the active countries can also make the process of searching for related research institutions easier by narrowing it down to certain locations.

RQ 1.6- What is the publication trend of SES papers by focus area and what are the most recent research areas?

Extracting the trends and recent research areas in the field gives the researchers in the related areas the possibility of directing their future research activities toward identifying and filling the research gaps and addressing the shortcomings of the field.

RQ 1.7- What are the activity levels of researchers from government, universities and private sectors?

This question aims at providing a general overview of the research activity breakdown in the field, based on the research group affiliation. The interested reader can have a broad view of activity levels originated from different research affiliations.

RQ 1.8- What is the trend of publications in academia, industry and government in SES research papers?

Similar to RQ 1.6, extracting the trends of the publications can potentially provide the reader with a general view of where to get affiliated with in order to be more actively involved in the field and to have a closer access to others working in the same field. Active research groups and laboratories can be identified more easily, when the interested researcher or grad student know where they are affiliated with.

RQ 1.9- What are the main goals of the papers?

The classification of the publications based on their goals in the review gives the readers and practitioners a glimpse of whether the issues of their interest are covered by the publications in the review or not. Also, together with the publication trends based on the focus area, the classification of the goals can be used for defining future research activities, where the related issues in the papers are mentioned unaddressed or need further investigations.

RQ 2- What are the main challenges in SES?

This question is mainly focused on pinpointing the challenges faced by domain experts, software engineers, scientists, or developers in developing SES. Looking from a software engineering mindset, software development has various phases often summarized as requirement elicitation, analysis, design, implementation, testing and finally maintenance. What makes the identification of the challenges more important in this study is that SES can be characterized by the particular challenges that researchers and practitioners might face during its development. Also the solutions provided can be used as a reference in similar context to deal with the complexities of SES development in real world. The sub-questions we raise are:

RQ 2.1- What are the challenges and solutions in requirements engineering and analysis of SES?

RQ 2.2- What are the challenges and solutions in the design stage of SES?

RQ 2.3- What are the challenges and solutions in the development (implementation) stage of SES?

RQ 2.4- What are the challenges and solutions in the testing of SES?

RQ 2.5- What are the challenges and solutions in the maintenance of SES?

RQ 2.6- What are the challenges and solutions in cooperation and human-related factors of scientific software projects?

RQ 3- What are the best practices in SES development?

This question aims at the identification of context-based best practices in the literature, that in particular are reported to make SES development a successful practice, while dealing with the challenges of developing software for scientific and engineering community.

3.3 **Study Selection Strategy**

The selection strategy we have used to choose the papers is described in the following sections.

3.3.1 *Source Selection and Search Keywords*

We searched through available online publications. A preliminary search (using "scientific and engineering software development" as the keyword) was performed in IEEE Xplore, ACM Digital Library and Google Scholar to extract all the related publications as well as to identify related online resources, e.g., well-known journals and conference proceedings in this area.

Based on the results from the primary search, another thorough search was performed which narrowed down the results to more specific publications, as explained below. The end result formed the database of all papers each of which in some way addressed our questions of interest. In order to ensure the completeness of this systematic review, we needed to make sure that we are including as many relevant publications as possible in the final pool of papers. To do so, we identified all potential search keywords regarding the focus of each of our research questions. We classify these strings mainly into two sets.

One set includes these major key words: "scientific software/application/program", "engineering software/application/program" "computing software/application/program", "scientific computing", "scientific computation", "computational science", and "scientific software development". The second keyword set contained more in-depth phrases, in addition to the above ones, e.g., "analysis", "maintenance", "requirement", "requirement elicitation", "design", "documentation", "implementation", "testing", "verification", "validation", "architecture", "risk", and "software engineering". Then the strings from the first group were combined with the strings from the second group to form a final set of comprehensive search key strings. By combining, we particularly mean using the AND and OR operator for concatenating our primary strings chosen from two sets respectively. Using these key strings, we also found a group of publications which were not relevant to our search questions, e.g. [60, 61].

We also specifically looked into several specific journals and the proceedings of several specific conferences and workshops in the area of SES, for example: the Elsevier journal on Advances in Engineering Software, IEEE Journal on Computing in Science & Engineering, Springer Journal of Scientific Computing, Workshop on Software Research and Climate Change and International Workshop on Software Engineering for Computational Science and Engineering.

3.3.2 *Study Selection Based on Inclusion and Exclusion Criteria*

We primarily chose papers based on their title, abstract and keywords. If not enough information could be found in the abstract, a careful review of paper contents was also conducted to ensure that all the papers had a direct relevance to different issues regarding the SES. We included journal papers, conference proceedings, theses, short papers and technical reports which addressed one or more of our research questions. To decrease the risk of missing related publications, we also looked through the references of the papers we found and included them if relevant. Inclusion process was not limited by defining any specific measure for quality, quantity or outcome of the research papers, therefore all the related empirical and theoretical papers were considered for inclusion. If multiple papers on the same topic and/or by the same author were found, the most recent one was included. Only papers written in English language and only the ones which were electronically available were included.

The publications which had no relationship to our research questions (the ones which did not discuss explicitly the development of SES in their research or practice) were excluded. For example, a large number of articles in venues such as the Elsevier Journal on Advances in Engineering Software are just presenting new software systems or algorithms for engineering purposes (e.g., [62]) and do not discuss issues related to the software engineering aspects of the software systems built. Such articles were excluded from this SLR.

Venue	# of Included Papers After Applying Search Query	# of Papers Left After Applying Exclusion Criteria
Springer Journals (e.g., Empirical Software Engineering, Lecture Notes in Computer Science, Engineering with Computers, Computer Supported Cooperative Work)	17	10
ACM Digital Library	51	33
IEEE Computer Society Digital Library (e.g., IEEE Software, Computing in Science & Engineering, Agile Development Conference, International Parallel and Distributed Processing Symposium, Symposium on Visual Languages and Human-Centric Computing)	39	23
Other venues (e.g., Elsevier Advances in Engineering Software, Briefings in Bioinformatics, Concurrency and Computation: Practice and Experience, PubMed)	57	17
Total	164	83

Table 3: Distribution of papers after applying inclusion and exclusion criteria

The author of this work together with her supervisor came up with the above mentioned inclusion/exclusion criteria. Based on these criteria, the author primarily decided about the inclusion/exclusion of the papers and then discussed with the supervisor to make a decision whenever there was an uncertainty about including or excluding a paper.

The earliest publication we found was from 1980 [63]. Since this study was conducted in 2010, articles published up to this year were included. Our pool initially had 164 related publications by applying the above search query. This number was reduced to 83 papers after applying the above mentioned exclusion criteria. Table 3 shows the details related to the distribution of the publications with respect to their publishers after the exclusion process.

3.4 **Data Extraction**

To extract data, the papers in our pool were reviewed by the author of this work and her supervisor and the information related to the research questions was extracted. The type of data and evidence collected from the papers related to each research question are summarized in Table 4.

In order to extract the above evidence from the studies, we needed to categorize primary studies based on the type of the evidence they presented regarding their research method. The breakdown of the publications based on their research method (inspired by [64]) is shown in Table 5.

As shown in Table 5, we grouped the primary studies based on their research method in several categories. In the empirical categories, we grouped all papers presenting case studies, field studies, surveys, experiments and reviews. We also grouped all the articles presenting experts' point of views and insights which were not explicitly supported by empirical evidence. We grouped together all the papers that presented related concepts or proposed a new idea, but the concept or idea was not empirically evaluated. Experience reports were also grouped together. It is worth mentioning that for all of the papers except the ones categorized under "concept implementation", the category was explicitly mentioned in the abstract or introduction of the papers. The interested reader can refer to appendix A, to find out more about the evidence type and main focus of each of the primary studies.

Research Questions	Sub-questions	Data Collected
RQ 1	RQ 1.1	Articles' publication year
	RQ 1.2	Application domains
	RQ 1.3	Software system size
	RQ 1.4	Programming languages used
	RQ 1.5	List of authors and the country where the research group is located
	RQ 1.6	The publication trend of SES papers per research goal and Recent SES research areas
	RQ 1.7	Research group affiliations
	RQ 1.8	Publication year and research affiliations
	RQ 1.9	Paper's main goals
RQ 2	RQ 2.1	Evidence from the studies which discuss requirements engineering and analysis
	RQ 2.2	Evidence from the studies which discuss software design
	RQ 2.3	Evidence from the studies which discuss software implementation
	RQ 2.4	Evidence from the studies which discuss software testing
	RQ 2.5	Evidence from the studies which discuss software maintenance
	RQ 2.6	Evidence from the studies which discuss human related aspects
RQ3		Evidence of best practices

Table 4: Data extracted for each research question

Evidence Type found in Publications	# of papers
Case study and Field study	16
Survey	6
Experiment	1
Review	1
Expert views without empirical backup	12
Concept implementation (proof of concept)	35
Experience report	12
Total	83

Table 5: Breakdown of primary studies based on the research methods

3.5 **Synthesis/Aggregation Method**

In this stage we needed to aggregate and synthesize the data extracted from primary studies. Synthesis is required to add more meaning and readability to the results. Since we had to deal with different types of evidence with various strength and creditability as the result of different research methods found in primary studies (as mentioned in more details in previous section), we selected narrative synthesis method [65]. In this method narrative description and explanation of the evidence taken from primary studies are presented along with commentary and interpretation [41].

There also exist other synthesis methods as mentioned in [41] such as meta-ethnography, thematic, and cross-case analysis. In meta-ethnography, primary studies are translated to one another either by approving each other, rejecting each other or building an argument line. In the thematic method, the recurrent themes in primary studies are identified and the findings are then summarized and presented under these themes. Finally in cross-case analysis, evidence is coded based on the identification of broad thematic headings and then presented by describing differences and commonalities.

We did not perform our synthesis based on the above techniques since the emphasis of this research was to extract all the related information from available literature to address our research questions, thus the focus of each research question was considered as a keyword or theme to search for information through the publications.

3.6 **Results**

We present in this section the findings of the review based on the questions we posed in Section 3.1. The results are presented for each of our research (sub-) questions.

We have done our mapping study in order to answer the sub-questions of RQ1. For RQ2, we classified the related papers into six groups (inspired by SDLC) as follows: papers discussing (1) requirements engineering, (2) design, (3) implementation, (4) testing, (5) maintenance, (6) cooperative and human related aspects. Each group provides data for one of the sub RQs. Since some of the papers discussed more than one stage in SDLC, these groups are not disjoint groups and there exist overlaps among the papers that are included in each group.

3.6.1 ***RQ 1- What are the demographics of studies in SES?***

In this section, we present the findings from our systematic mapping study. We focused on demographic characteristics which reflect the variety and frequency of the research on different areas and domains for which SES are developed.

3.6.1.1 RQ 1.1: What is the trend of the publications over years?

We counted the number of publications in each year. Publication's trend from 1980 to 2009 is shown in Figure 4. The publication year of the earliest paper [63] in our pool was 1980 which proposed new techniques that can be applied to complex real-time flight software systems for requirements specification of the software systems built for the US Navy's A-7 aircraft.

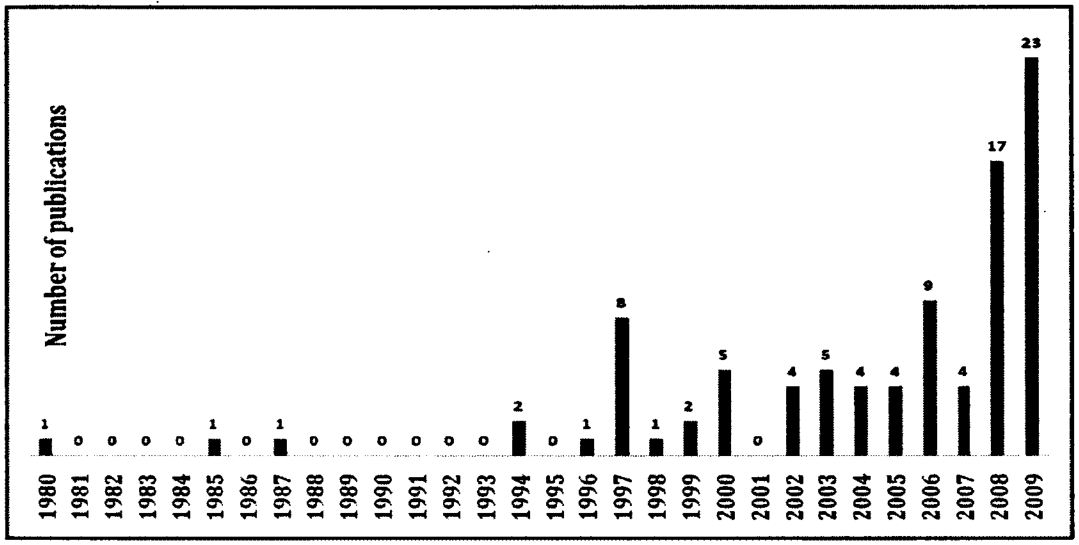


Figure 4: Number of the publications between 1980 and 2010

As we can see in Figure 4, there were very few papers until 1995. The number of the publications starts to rise sharply in 1997. Our pool does not have any paper published in 2001. We had most of the paper publications in 2008 (17 papers) and 2009 (23 papers). Also, we notice that the cumulative number of the publications has almost doubled from 2006 (48 papers) to 2009 (81 papers).

3.6.1.2 RQ 1.2: What are the application domains?

We categorized the publications with respect to their application domains. The papers distribution is shown in Table 6.

Based on the table, 41% of the papers did not mention any specific domain for their study. About 15 % of the papers (mostly survey and case study papers) have conducted research on several disciplines. Among other domains, physics and biology were the most common domains.

Reference Discipline	Percentage of papers
Automotive industry	1.2 %
Agriculture	1.2 %
Weather forecasting	2.4 %
Mathematics	2.4 %
Chemistry	3.6 %
Aerospace engineering	3.6 %
Image processing	6 %
Biology	8.4 %
Physics	14.5 %
Mixed disciplines	15.7 %
Not mentioned	41 %

Table 6: Publications Application Domain

3.6.1.3 RQ 1.3: What is the size of software systems under study?

Not all the papers revealed detail information regarding size or complexity of the software systems under study. However, about a dozen of the papers did mention that information. The information we found in other papers about the size of their projects is summarized in Table 7.

Name of the application	Description	Lines of code (LOC)
Java Imaging Utilities [66]	API for image manipulation	~43,000
Art of Illusion [66]	3D model and image rendering	~65,000
UM (Unified Model) code base [10]	A code suite for numerical weather prediction and climate models	~830,000
Kahindu Medium [66]	Tool for image manipulation using image filters	~108,000
Osprey [29]	A component of a large weather forecasting suite	~150,000
FALCON [23]	Product performance evaluation software	~405,000
HAWK [23]	Manufacturing process analysis software	~134,000
CONDOR [23]	Product performance simulator	~200,000
EAGLE [23]	Signal processing software	Less than 100,000
NENE [23]	Molecule modeling software	~750,000
MI [67]	OO environment for integration of scientific applications	~140,000
CC1, CC2, CC3, CC4, CC5 [36]	Computational Chemistry software	10,000...320,000
PMGT [68]	Parallel Mesh Generation Toolbox	~3,000
BlobFlow [69]	Two dimensional Navier-Stokes equations solver	~10,000
~:Approximately		

Table 7: Application sizes in LOC

Hochstein et al. [24] mentioned that they are working with a group of projects for computational simulation (e.g. on solid mechanics, fluid mechanics, combustion)whose size is between 100-500 KLOC. Easterbrook et al. [10] presented a graph on the growth of their project over 15 years where they showed the project size was about 110 KLOC in 1993 and grew to around 830 KLOC in 2009. Kelly et al. [70] conducted a study assessing the quality of 10 SES whose size varied between 1 to 100 KLOC.

As seen in the above table, PMGT [25], which is a parallel mesh generation toolbox is the smallest tool in our pool with 3 KLOC. UM (Unified Model) system [10] which is a software for numerical weather prediction with 830 KLOC is the largest project among the others.

3.6.1.4 RQ 1.4: Which programming languages are being used?

In our paper pool, we had 72 papers mentioning the programming language used in their projects. As shown in Figure 5, the majority of the applications were written in Fortran. The most widely used versions of Fortran, as mentioned explicitly in seven papers out of a total of 24 papers, were versions 77 and 90. In nine papers, Fortran was being used together with a second (or more) language, e.g., C++. After Fortran, Java, C++ and also Python are the most widely used ones. Matlab and Perl come next, while the use of Ruby and PHP was only reported in one paper.

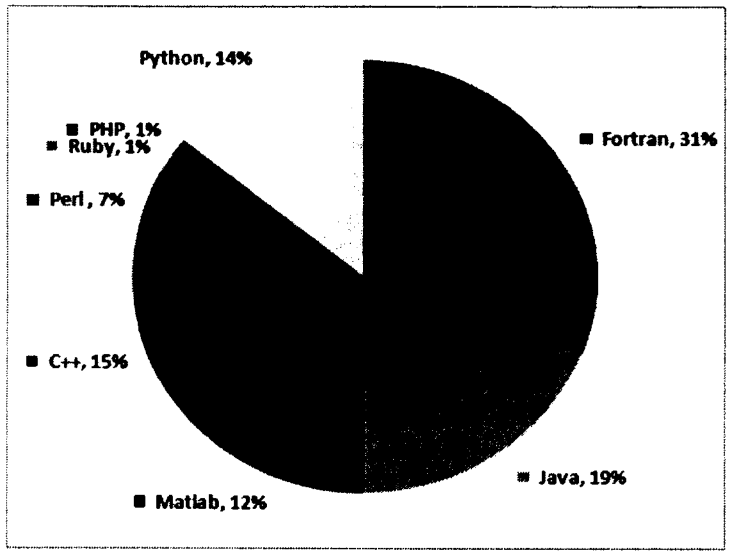


Figure 5: Programming language distribution

In a study by Carver et al. [23], the authors reported five case studies of SES development projects and each project was from a different scientific or engineering domain. They found out that the primary language of the projects does not changes over time, meaning that Fortran remained the dominant language in the projects and the use of high-level and object-oriented languages (such as Java or C++) were relatively low.

In a paper by Cary et al. [71], a comparison between Fortran 90 and C++ was undertaken when being used for OO scientific programming. C++ is identified to have full support for inheritance and polymorphism while Fortran 90 does not support inheritance. It has been shown that Fortran 90 specific features such as mathematical arrays and specifiable precision of floating-point numbers can be added to C++ by the implementation of class libraries.

To support portability and reusability, C++ supports templates which are not supported in Fortran 90. Though primary C++ compilers used to be slower than Fortran

[72], optimized C++ compilers can be used now to generate executable programs that can compete with those generated by Fortran compliers in terms of performance.

As discussed by Veldhuizen et al. [72], there is no more a need for mixed-language programming where the framework of the programs were written in C++ and the routines which needed to have high performance were written in Fortran. In another article by Veldhuizen [73] the performance of C++ in comparison to Fortran is shown to be practically better (i.e., higher performance) using different known benchmarks (frameworks to assess the performance) in linear algebra, and array stencilling.

We tracked the trend of Fortran, Java and C++ as the most popular languages over the publication year. The cumulative numbers of papers reporting on usage of Fortran, C++ or Java as their choice of programming languages are visualized in Figure 6. Although it is not easy to draw clear conclusions on the increasing or decreasing popularity of any of these languages, but we can see than the use of Java as their choice of programming languages by SES developers is growing faster than that of C++.

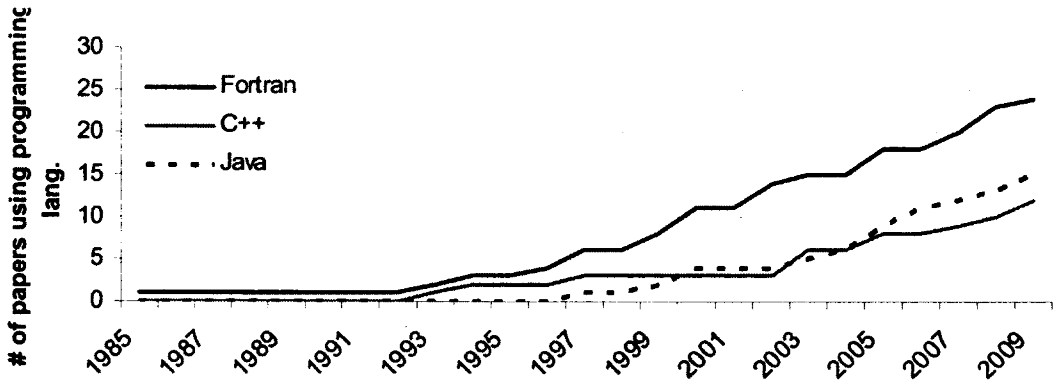


Figure 6: Cumulative number of papers reporting on usage of Fortran, C++ or Java as their primary choice of programming languages

Also according to the survey by Tang [26] C, Fortran and C++ were the most popular programming language used in SES development. The use of other languages such as C# and VB.Net is also indicated to be popular among the respondents of the survey.

3.6.1.5 RQ 1.5: Which researchers and countries have been more active?

To conduct a bibliometric analysis to rank active researchers and countries, we counted the number of papers published by each author in our pool of papers. Results for the researchers who have at least two or more publication in our paper pool are shown in Figure 7.

Diane Kelly, affiliated with the Royal Military College of Canada [3, 70, 74-78] and Judith Segal from Open University in the UK [6, 8, 20, 33, 79-81] with seven papers have the lead among all the authors in our pool. Douglas Post, Richard Kendall and Jeffrey Carver with five papers stand next.

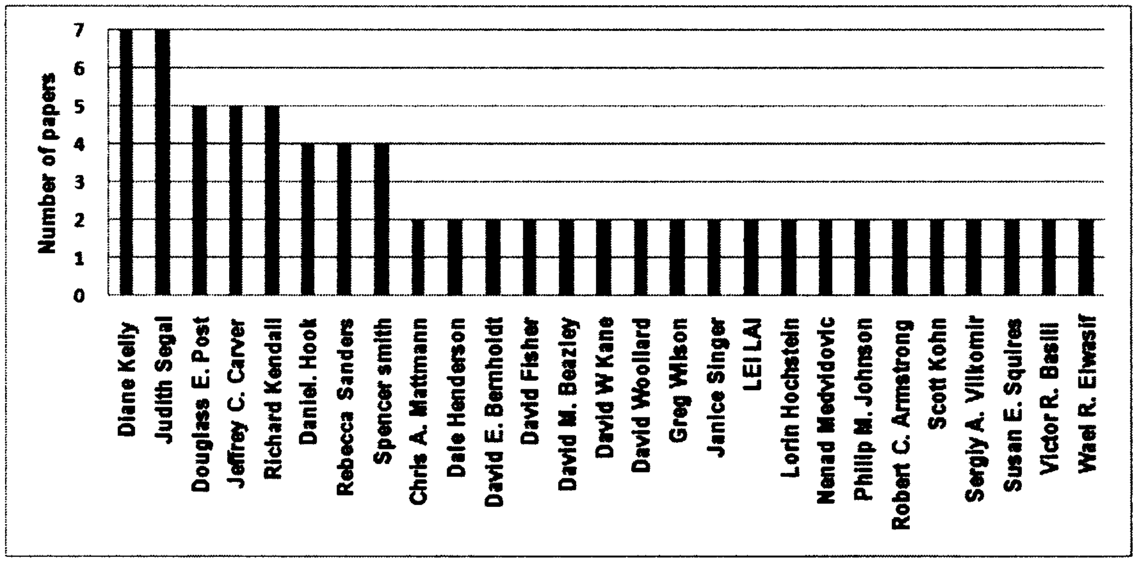


Figure 7: The most active authors

We also wanted to see which countries are more active in this area. The distribution of countries based on the author affiliations is shown in Figure 8. If a paper had several authors from two or more countries, we added a weight to each of those countries (sum of the weights for each such paper being equal to 1).

As seen in the figure, USA, Canada and the UK are the top 3 contributing counties to the literature on SES. Only 18 countries have published papers in this field. Except Australia, Brazil, India and Japan, all of the other 14 countries are from North America or Europe.

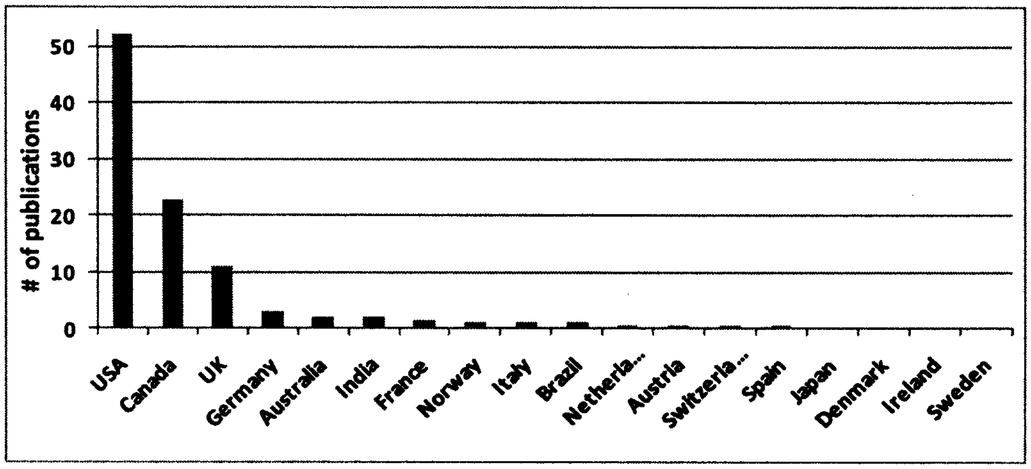


Figure 8: Active countries in publishing research papers on SES

3.6.1.6 RQ 1.6: What is the publication trend of SES papers by focus area and what are the most recent research areas?

To analyze the publication trend of SES papers, we used the same categorization as in Table 9 and counted the number of the papers published in each year.

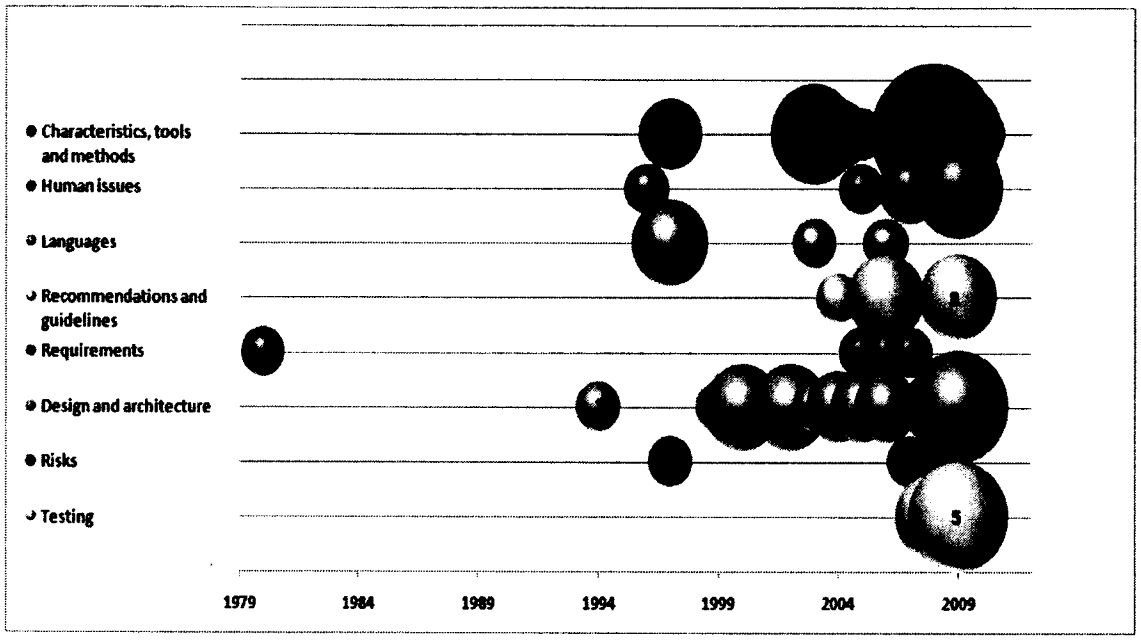


Figure 9: Timeline per publication focus area

Results are shown in Figure 9 using a bubble chart. The size of the bubbles in each category per year is proportional to the number of the publications (as marked in the figure with numbers 1, 3 and 5 as samples) for that category in that year.

As it is shown in Figure 9, most of the recent publications in our pool were about testing, design and architecture and human related issues in SES development as well as the papers on characteristics, methodologies and tools. This shows these areas have gained more attention in the field but other areas such as requirement elicitation and maintenance still need further investigations.

The most recent research areas within the last 5 years (2006-2010) were also extracted and shown in Table 8 to give the reader a detailed view of the recent publication's topic in the field.

Year	Topics	References
2010	SE in Embedded and mobile robot software	[49]
	Code development for computational Chemistry	[82]
2009	SE for climate change	[10]
	Bioinformatics Software development	[35]
	Scalable software development	[36]
	Developing high quality Parallel Mesh Generation Toolbox	[68]
	Testing SS	[75, 76, 83-85]
	SES development characteristics, practices and problems	[26, 50, 74, 80, 81, 86-88]
	SS design	[89-92]
	SS development by generative programming	[93]
2008	SES development characteristics, practices and problems	[4, 6, 8, 33, 51, 94-96]
	Developing weather forecasting code	[29]
	High-performance computing	[34]
	SS testing, verification and quality assessment	[69, 70, 77, 97]
	Large-scale parallel scientific code development	[24]
	Dealing with risk in SS	[78]
	Developing quantum chemistry science application	[98]
	Managing SS complexity	[99]
2007	SES development characteristics, practices and problems	[3, 20]
	Development environments	[23]
	High-performance computing characteristics and risks	[100, 101]
	Requirement analysis	[102]
2006	SES development characteristics, practices and problems	[5, 103]
	Design complexity	[66]
	Development of requirement documentation	[25]
	Agile methods in biomedical software development	[104]
	Challenges in automotive software engineering	[105]
	Language interoperability issues	[106]

Table 8: Recent research topics in SES development

3.6.1.7 RQ 1.7: What are the most active sectors for software systems under study? (e.g., government, universities, private sectors)

By extracting the research group affiliation information from the papers, we categorized them into four groups: (1) governmental research groups such as NASA, (2) the ones affiliated with universities, (3) corporate research groups and laboratories, and (4) collaborative work (a combination of two or three other sectors). The results related to this classification are shown in Figure 10. Not surprisingly, majority of the works have been published by university labs and research institutions affiliated with universities.

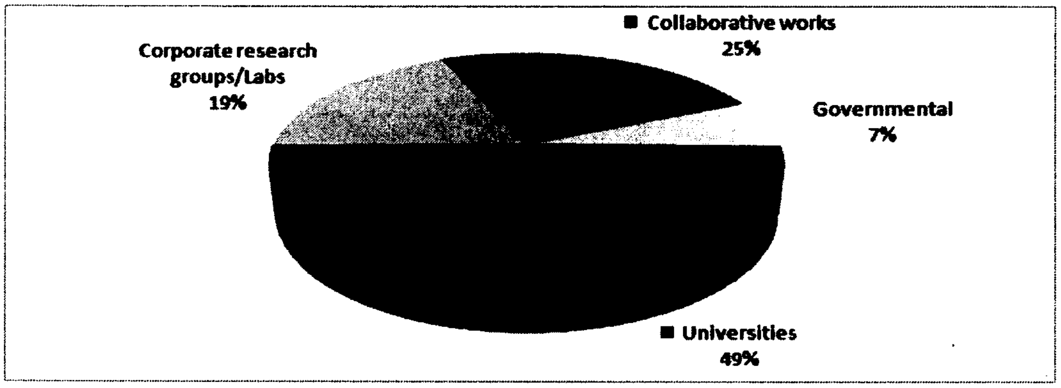


Figure 10: Classification of research group affiliations

3.6.1.8 RQ 1.8: What is the trend of publications in academia, industry and government in SES research papers?

We gathered the number of the publications of each research sector over the publication year in order to study the publication trends for each sector. The results are depicted in Figure 11.

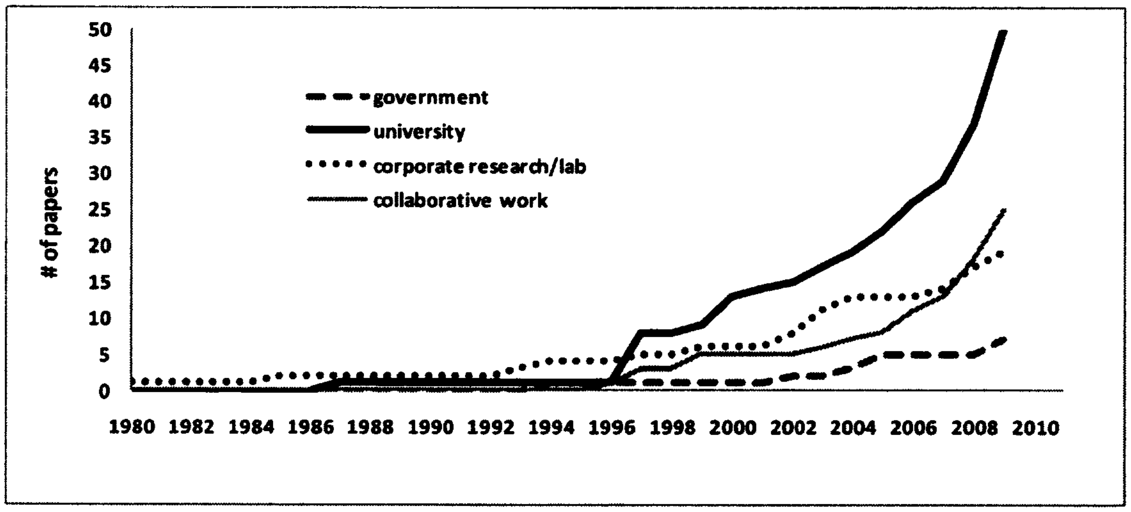


Figure 11: Cumulative trend of publications in different research sectors

As shown in the figure, the number of publications in each of the four categories follows an increasing trend over years. In the academic section, which surpasses other sectors significantly, the growth is noticeably quicker. Corporate research groups/labs have started publishing papers in the field way ahead of the other groups. The publications in intersection collaborations are also increasing rapidly in recent years.

3.6.1.9 RQ 1.9: What are the main goals of the papers?

The publications were categorized based on their main goal, in a way that categories have the least possible overlap with each other. We came up with seven different categories as shown in Table 9. The majority of the papers fall under the category named "characteristics, methodologies, tool and environments". In this group, the papers mainly discuss different approaches (e.g., using algebra systems, problem solving environments,), methodologies (such as Agile) as applied to the development of SES as well as characteristics of SES (such as high-performance computing systems).

Main topic of the paper	**Paper references**	**# of papers**
Characteristics, development methodologies, tools and environments	[7, 10, 23, 24, 26, 29, 31, 33, 34, 68, 82, 86, 87, 94-96, 100, 104, 107-113]	25
Issues and challenges related to different types of developers and their attitudes	[3, 8, 20, 35, 50, 79-81, 114]	9
Requirements	[25, 63, 102, 115]	4
Design and architecture	[66, 67, 89-93, 98, 99, 105, 116-127]	22
Testing SES	[70, 75-77, 83-85, 97]	8
Different types of risks	[78, 101, 128]	3
Lessons learned, guidelines and recommendations for SES development	[5, 36, 51, 74, 88, 103, 129]	7
Languages used in the development	[71, 72, 106, 130, 131]	5

Table 9: Classification of the primary studies based on their main goals

The second category called "issues and challenges related to different types of developers and their attitudes" mostly includes the papers which discuss issues originated from the differences between scientists and software engineers. "Lessons learned, guidelines and recommendations" group consists of the papers discussing the experiences of the authors and their proposed best practices. The rest of the groups' names are self-explanatory as shown in Table 9.

3.6.2 ***RQ 2- What are the Main Challenges and Solutions in SES?***

We investigate the challenges, solutions and other observations in further detail next through RQ 2.1-RQ 2.6, which focus on each SDLC phase.

3.6.2.1 RQ 2.1: What are the challenges and solutions in requirements engineering of scientific software?

In order to provide the reader with the type of evidence we had in each section, we counted the number of different papers in each category of evidence. We had 16 papers mentioning the issues related to the requirements in SES development. Among them, there are three case studies, one field study, one experience report, two surveys, three concept implementation and six expert views. Table 10, provides information on these papers main focus, evidence type and context besides a brief description of the challenges, solutions and observation reported in each papers. 'General', under the 'Context/Domain' column in the table means no particular context was mentioned in the papers, or the authors explicitly claimed that their findings are applicable to the broad context of SES development. The "Specific to SES" column shows whether the challenges presented in the literature are specific to SES development or they are common concerns of any type of software development practice, and whether the solutions suggested here are also being adopted elsewhere or not.

Ref	Paper's main focus	Evidence type	Challenges	Solutions/Observations	Specific to SES	Context/ Domain
[95]	SS development experiences and practices	Exp. report	Not being able to fix the requirements when there exists no solution for the scientific/engineering problem	Solution: starting the development with estimation of basic requirements	Challenge: Yes Solution : No	Developing control and data acquisition software
[79]	Investigating the case where software engineers developing software for research scientists, using a traditional, staged, document-led methodology	Case study			Yes	Space scientific software development
[100]	SE for high performance computing	Expert view		Observation: the requirements conform to mathematical models	No	High performance systems
[33]	Improving SS development			Solutions: engaging scientists in software development	Yes	General
			-Gaining a correct and precise understanding of the problem domain and application requirements -Complexity of requirement specification -Complexity of requirement elicitation		Yes	
[23]	Identification of the steps and tools in developing high-performance software	Case study			Yes	High performance computing
[25]	Presenting a methodology for development of the requirements	Concept implementatio		Solutions: writing the requirements in a testable way	No	General

			Difficulty in prioritizing non-functional requirements		No	
[80]	Challenges of software engineers	Field study	Gathering of the requirements by scientists		Yes	Software engineers developing software for space scientists and biologists
[68]	Proposing a methodology based on software requirement specification	Case Study		Observation: use of the requirement as a contract between developers and testers to promote verifiability	No	Mathematics
[63]	Presenting new techniques for making requirements specifications precise, concise, unambiguous, and easy to check	Con. Implem.	Not having a reliable resource for further reference, when facing a conflict between software engineers and domain experts	Solutions: Documenting the confirmed requirements between the developers and domain experts at each stage	No	Flight software development
[115]	Proposing a new template for requirement specification		Difficulty of validating the requirements		No	General
[113]	SE for high performance computing	Expert view		Observation: the requirements conform to mathematical models	No	High performance systems
[112]	SE for high performance computing				No	High performance systems
[105]	Developing software for automotive industry		Dealing with innovative and modern requirements		No	Automotive software development
[101]	Identifying different risks in high performance computing applications		Dealing with risks in requirement engineering		No	High performance computing

[50]	Surveying how scientists develop and use SS	Survey		Observations: enough attention is paid to requirement elicitation when the development team is large	No	General
[26]	Developing scientific and computing software			Observation: use of informal specifications is more common- correctness and reliability are found to be the most important non-functional requirements	No	General

Table 10: Summary of the papers discussing requirement issues

In the following sections more details on the challenges, solution and other observations concerning the requirement in SES are presented.

Challenges

For a developer who is not an expert in the scientific/engineering field of a to-be-developed software, gaining a correct and precise understanding about the problem domain and the application requirements is the very first challenge [33]. As reported by Segal in [80]: "scientists may not appreciate that the gathering of requirements at both the high (functional) and low (user) level is often a significant part of software development".

Another characteristic that tends to be problematic is that, in most of SES projects, the requirement cannot be fixed and finalized in the early stages as reported by Segal in [33]. Specially, while developing software to discover a scientific or engineering problem for which there exists no prior solution, it is very challenging to fix the requirements upfront [23, 100].

Requirement elicitation, though being very important, is often neglected in scientific software development according to Smith [25]. This is in particular problematic when in the design or implementation phase, ambiguities begin to emerge and there exists no criterion on how to resolve them easily and properly. On the other hand, as mentioned before one specific characteristic of requirements in SES is that usually they cannot get finalized at the beginning of software development as stated in the experience paper reported by Segal [79].

In most scientific applications, the requirements specification is very difficult to validate because the quantities are often continuous in comparison to other typical commercial software where values are discrete[115].

In the domain of automotive systems, a "fitting requirements engineering method" is stated to be a big problem [105], as most of the requirements are innovative and modern.

In terms of non-functional requirements, especially performance, usability and portability, building a system with a realistic and feasible trade-off is another great challenge. Smith discusses in [25] that it is often not trivial to give a valid priority to one non-functional requirement factor over another.

Kendall et al. in their study [101] identified the risks of the requirement phase of high performance computing applications such as unpredictability of the requirements, failure to address the constant evolution of the requirements and having incomplete, unclear and inaccurate requirements.

Possible solutions

According to another paper by Segal et al. [33], the nature of requirements in the case of SES often leads to the above challenges, unless the scientists themselves are entirely in charge of software development. Otherwise the developers must reach to a good understanding of the domain before starting the development, which is not a trivial task.

After validating their understanding, the developers should be committed to document what is already confirmed between them and the domain experts to build a reliable resource for further references, as suggested by Heninger [63].

Smith [25] suggested that the requirements specification should be written in a way that it is testable and easy to validate. According to Achroyd et al. [95], one of the characteristics of a successful scientific software development project is that it will start with an estimation of the basic requirements and later, as all parties learn how to cooperate efficiently, detailed requirements would be added to support extra functionality. The authors highly recommend not putting too much demand on the domain experts to finalize the requirements upfront.

Other observations

According to a recent survey by Hannay et al. [50], developers of SES working in large teams are more likely to pay enough attention to requirement elicitation and analyzing rather than the developers in small teams and the ones who are working on small projects.

In high performance systems as stated by Carver [100] and Johnson [112, 113], the requirements often must conform to sophisticated mathematical models and can be an executable model in a system such as Mathematica.

Smith in [68], discussed about the critical role of requirements to quality of the software and mentioned that "software requirements serve as a contract between developers and testers; therefore, the SRS (software requirements specification) promotes verifiability by giving the testers something to verify against."

According to Tang's thesis [26], 70 percent of the respondents to his survey were using informal specifications for their project requirements. Among non-functional requirements, correctness and reliability are rated higher as the respondents believed that the quality of the software highly relies on them. On the other hand, security and memory usage were the least considered non-functional requirements.

To provide empirical evidence on the above issues from our own experience based on our meeting minutes with our industrial partners, we report next the actual challenges we have been experiencing in our ongoing major optimization software development project for oil pipelines' pump operation [132]. When the project started in early 2008, the team became involved in requirements engineering and analysis of the system.

The team was composed of three software engineers (one of whom had good knowledge of optimization techniques), one civil engineer also with good knowledge of optimization techniques, and a mechanical engineer (as domain expert) from an industrial firm in Alberta, Canada. Although the final product of the requirements engineering phase was of good quality, the team had numerous challenges along the process, e.g., finding a consistent vocabulary to understand one another, prioritization of major features versus minor ones and deciding on the interoperability requirements of the to-be-built system with existing software systems used by the industrial partner. One of the software engineers remembers many occasions in which he was struggling to use less-technical software engineering and optimization vocabulary to be able to smoothly communicate with the domain experts (i.e., the mechanical engineers).

3.6.2.2 RQ 2.2: What are the challenges and solutions in the design phase of scientific software?

We discovered 21 papers mentioning the issues of design in SES development. The breakdown of the evidence type of these papers is as follows: two surveys, two case studies, one experience report, 13 concept implementations, one experience/interview, one case study/survey and one expert view. Table 11 summarizes these papers main focus, context/domain and the reported challenges, solutions and observation regarding the design in SES development.

Ref	Paper's main focus	Evidence type	Challenges	Solutions/ Observations	Specific to SES	Context/ Domain
[117]	Investigation of the risks in a modeling framework and how to address them	Experiences and interviews	-Unrealistic user expectations - Premature obsolescence of the design		No	Developing a framework for assessment of how future alternative agricultural and environmental polices affect sustainable development in Europe
[34]	Characterizing high-performance computing community	Case study and survey	Not having background to apply OO principles		No	High-Performance-Computing

[101]	Identifying different risks	Expert view	Having complex requirements makes the design complex		No	High performanc e computing
[121]	Using OO technology for the design of satellite data processing software	Concept implementation	The complexities of modeling: physical (problem at hand), mathematical (formulation) and software (practical solution) modeling		Yes	Satellite data processing software
[125]	Investigating the incorporation of message passing systems into component-based systems		Incorporating componentized message-passing libraries in a parallel/distributed environment		No	High-performanc e scientific computing
[116]	Introduction on using patterns for SS		Incorporating reusability and maintainability	Solution: using design patterns	No	Dynamic-systems simulation
[119]	Presenting design patterns for SS and explaining their benefits				No	Computatio nal life sciences
[111]	Proposing a framework for multi-physics simulations		Abstracting and managing data and functions in their modules	Solution: using OO technology to manage complexity and to support reusability	No	Multi-physics simulation
[67]	Integrating scientific applications		Software reusability		No	Builidng scientifc software models
[127]	Presenting the Common Component Architecture for managing the complexity in high-performance scientific computing			Solution: using component-based technology to support reusability and integration	No	High end scientific computing

[126]	Developing SS component technology		Interface design-integrating code from different programming languages		No	High-performance scientific simulation
[99]	Introducing a tool to perform rapid component prototyping while maintaining robust software engineering practices		The complexity of learning the details of the component interface, while using component-based technology		No	High-performance scientific computing
[93]	Generative programming for SS developments		Difficulty of creating domain-specific solutions from reusable software components	Solution: using generative programming approach	No	Image retrieval-poison solver
[92]	Integrating architectural constraints with legacy SS		Integrating legacy systems with modern systems	Solution: use of architecturally-aware interfaces to wrap the scientific code of the legacy systems to integrate them with modern systems	No	Dealing with legacy scientific code
[124]	Proposing a standard to support interoperability among high-performance scientific components			Observation: use of common component architecture	No	High-performance scientific computing
[91]	Proposing a framework to involve the domain experts in design			Observation: engaging domain experts in design were found successful	No	General
[31]	Details of developing software for computational biology and bioinformatics	Experience report		Observation: well-design reduce data complexity, ease access to modeling tools and support integrated access to diverse data resources	No	Computational biology and bioinformatics

[66]	Investigating the complexity of design	Case study		Observation: the complexity of a software system is very dependent on the design knowledge of the developer	No	Scientific imaging software
[89]	Usability and user-centered designcase study		Managing user expectation in user-centered design		No	Imaging software developme nt
[50]	Surveying how scientists develop and use SS	Survey		Observation: SE practices are more commonly used for larger project and teams	No	General
[26]	Developing scientific and computing software			Observation: system design specification and detailed design specifications were provided by the designers	No	General

Table 11: Summary of the papers discussing design issues

In the following sections more details on the challenges, solution and other observations identified by our survey concerning the design in SES are presented.

Challenges

According to Gupta et al. [121], the design of a good quality SES needs to tackle three different modeling challenges. First, the physical modeling in which the phenomenon and its underlying basics must be understood. Second, the mathematical modeling, which is the process of formulating physical models, and finally the software model which refers to the practical solution for the problem inspired by previously built physical and mathematical models.

According to the experience of Basili et al. [34], many scientists do not have enough background in object-orientated analysis and design and, subsequently, are not very skilful in developing complex object-oriented programs.

Bernholdt et al. in [125] identified the issue of incorporating componentized message-passing libraries in a parallel/distributed environment, which needs major modification of application code and may lead into runtime overhead.

The authors in [117] described the trade-offs in the design of their modeling framework. They identified "unrealistic user expectations" (or business goals) and "premature obsolescence" as the main challenges in their design process. They also reported the difficulty of "incorporating those architectural aspects in the design which do not comply with the business logic".

Kendall et al. in their study [101] identified the risks of the design phase of the high performance computing applications as having difficult requirements and expectations, and the need for a design which supports modularity, maintainability, portability, reliability and usability in general.

Possible solutions

The use of design patterns in large SES software development projects have shown remarkable benefits toward adding more reliability, reusability and better maintenance as reported by Blilie in [116]. He argued that this is only possible by introducing the concept of object-oriented design which has its own pros and cons in the context of SES. Object-oriented languages (e.g., Java and C++) are known to have higher computational overhead compared to procedural languages (such as Fortran) while, most of the time, performance and speed are important attributes of a successful end artefact. That is one of the reasons why most of the current scientific software have been developed in C and Fortran as reported in [34, 116].

In another paper by Cickovski et al. [119], again the benefits of using design patterns such as speed, memory consumption, flexibility, and software maintenance were mentioned.

Authors in [111] described the use of object-oriented technology for abstracting and managing the data and functions in their modules. As the size of projects continues to increase, the benefits of using object-oriented design can outweigh its drawbacks because the code will become more and more complex to manage. In order to support the reusability of the code and also in order to integrate the code and tools from different disciplines, more and more SES developers are adopting the object-oriented technology, according to the experience of Spinelli et al. [67].

According to Bernholdt et al. [127], using object-oriented methodologies can lead to a robust framework for different libraries, where components can be fitted and used toward better managing different parts of the system. Components are reusable software packages which embody a group of useful functions. Component technology tries to resolve major issues in software reuse and integration such as barriers in interface design, physical deployment and integrating code from different programming languages mostly by removing the language and compiler dependencies as reported in [126] by Epperly et al. However looking from another perspective, employing component-based technology for SES design can add to the complexity, as the user of the component needs to learn the details of interfaces for managing the systems interactions and the conventions of the component model as reported by Allan et al. in [99].

Arora et al. in their paper [93] described the generative programming approach for developing SES. In this approach the desired software system can be automatically built from the given specifications and domain-specific solutions can be created from reusable software components. The approach has been shown to increase the level of abstraction while decreasing development time and costs.

Woollard et al. in their paper [92] discussed the benefits of software architectures and proposed the use of architecturally-aware interfaces to wrap the scientific code of the legacy systems in order to integrate them with modern systems.

Other observations

According to Gentleman et al. [31] "well-designed scientific software should reduce data complexity, ease access to modeling tools and support integrated access to diverse data resources at a variety of levels."

The complexity of a software system is more dependent on the design knowledge of the developers rather than the application domain or the type of the system that is being developed as reported by Larsson and Laplante in [66].

In this phase, again as for the implementation, employing software engineering practices will gain its attention as the project and team size grow according to the survey by Hannay et al. [50]. Thus in small size projects, to come up with a systematic and robust system design remains a challenge.

Macaulay et al. [89] investigated the design usability and user-centered design in their project called Usable Image. For that purpose, they tried to investigate the details related to their use environment and to increase their user-base to contain all possible users even outside their labs. They experienced the challenge of managing user expectations as in user-centered design the user will get used to seeing prompt responses to their feedbacks.

Armstrong et al. [124] proposed a CCA (Common Component Architecture) for developing high-performance SS as in "high resolution and complex physical sub-models for turbulence, chemistry, and multiphase flows". They developed a single component interface specification for supporting the interactions among scientific components. The architecture consist of a SIDL (Scientific Interface Definition Language) to describe the interfaces, CCA ports which defines the communication model for component interactions and CCA services which is a framework abstraction.

According to Tang's thesis [26], 45% and 27% of respondents to his survey, mentioned that they have system design specification and detailed design specifications, respectively. Software reuse reported to be very popular among the developers as only an insignificant of the respondents indicated that they are not reusing their software.

In terms of human aspects of design, putting the domain experts in charge of designing their own product by providing them with the required tools and techniques is shown to be a success factor by Fischer et al. [91]. Such a framework is called "meta-design" by the authors of [91]. In this framework, different techniques and methodologies are encompassed to give domain experts the freedom of acting as a designer by being involved with the development process rather that just limiting their role as the end-users of the system. The human-problem interaction is supported while the focus is not on building the final solutions. The users/developers are provided with a space in which they can build their specific solutions to fit their own needs.

3.6.2.3 RQ 2.3: What are the challenges and solutions in the implementation phase of scientific software?

We had 14 papers mentioning the issues of implementation and coding in SES development. Among the papers we had two surveys, seven concept implementation, three experience report and two expert views. Table12 summarizes the information extracted from these papers.

Ref	Paper's main focus	Evidence type	Challenges	Solutions/ Observations	Specific to SES	Context/ Domain
[77]	Identifying different types of risks in testing SS development	Survey	Correctness of the implementation- poor code documentation		No	A mixture of engineering and scientific disciplines
[26]	Developing scientific and computing software			Observation: industry is much more careful compared to academia in terms of coding standards	No	General
[106]	Proposing an approach to fill the language gap in SS	Concept implementation	Language interoperability		No	Langauge interoperability
[111]	Proposing a framework for multi-physics simulations				No	Multi-physics simulation
			Concurrent code implementation, check-pointing		No	
[122]	Proposing the use of a compiler to automatically optimize software library implementations		Limitation of software libraries	Solution: using simple declarative annotation language that describes certain aspects of a library's implementation to optimize the use of the libraries	No	Scientifc library implementation
[98]	Component-based architecture in quantum chemistry SC		Implementing and adopting uniform interfaces in component-based architecture- managing software dependencies and build systems	Solution: building a generic package that enclosed uniform interfaces to manage software dependencies	No	Quantom chemistry application development

[123]	Proposing a new architecture for SC application development			Observation: proposing an infrastructure that provides the user with an easy programming model and API and incorporates different types of computational modules	No	General
[110]	Using computer algebra systems to automatically generate a computer program			Observation: using computer algebra systems	No	Generating scietific code
[130]	Automatic SS scripting			Observation: developing an extensible compiler to automate the integration of compiled code with scripting language interpreters	No	Large-scale parallel molecular dynamics simulations
[96]	Automating scientific workflow	Experience report	Software integration- managing different scientific activities	Solution: use of workflow management systems	Yes	Developing scientific workflow manageme nt system for collecting, analyzing, and managing data produced by sensors and other instruments
[94]	Proposing and characterizing workflow systems		Assembling scientific code into an executable system		Yes	Parallel computatio n over data sets
[131]	Using python in SS development		Managing huge amount of data- dealing with frequent software changes	Solution: addressing implementation problems by using Python	No	Large-scale physics application
[109]	Problem solving environments	Expert view	Difficulty of making physical simulations reliable	Solution: using problem solving environments	No	General

[105]	Developing software for automotive industry			Observation: highly optimized code makes reuse and maintenance quite hard	No	Automotive software development

Table12: Summary of the papers discussing implementation issues

In the following sections more details on the challenges, solution and other observations concerning the implementation of SES are presented.

Challenges

The implementation phase of SES development is also challenging as certain types of risks can be identified in this stage according to Sanders and Kelly [77]. The authors have identified three types of risks related to code which makes testing more difficult. The first one is risk to correctness and is mainly concerned with the accuracy of the calculations, which is a very critical quality factor. The second one is the risk from poor code documentation, which was identified to be very common in ESS development. The last one is risk to verification meaning that we need to ensure that the code solves the models or the desired equations in a right way. This last risk is a major problem as the scientists usually do not know how to test their code or they are even unaware of the need for that.

Also the problem of language interoperability is an issue that happens when the developer wants to merge the core of existing scientific software with the software tools which are mostly written in high level languages such as C++, Java or C# [106].

In a paper by Jiao et al. [111], different challenges of implementing a large-scale numeric software called Roccom for multi-physics simulations were mentioned. Issues such as concurrent development of different modules, programming language interoperability, complexity of coupling schemes, check-pointing and plug-and-play capability are discussed and then object-oriented design and architecture of the system is presented.

Guyer et al. [122] in their paper described the weaknesses and performance limitations of software libraries such as not being able to use the library implementation, which suites the needs of a particular client. On the other hand, making the library generalizable reduces its efficiency.

Kenny et al. in [98] described the challenge of implementing and adopting uniform interfaces in component-based architecture to enable interchangeability and interoperability among different packages. They mentioned "managing software dependencies and build systems" as another challenge of large-scale systems.

Possible solutions

Some tools and packages have already been developed to resolve the challenges of language interoperability, e.g., Chasm [106] and CLI [133], but still there exists room for further investigations and studies in this area.

Guyer et al. in their paper [122], explained how, by using "a simple declarative annotation language that describes certain aspects of a library's implementation" the libraries can be used in an optimized way.

Kenny et al. in [98], addressed the issue of managing software dependencies by building a generic package, which enclosed uniform interfaces and by creating a library which had the glue code to access the interfaces in the supported programming languages.

The use of workflow systems in order to support the scientist's work and address the challenges of developing SES were discussed in some papers [94, 96]. Vidger [96] defined a workflow system as "commonly used, well-defined sequences of data manipulation procedures, which involved activities such as numeric transformations, format changes, analysis, and file management". Woollard et al. [94] proposed the use of workflow environments for a key activity termed "orchestration" which is explained as "assembling scientific code into an executable system with which to experiment". In their paper they also discussed the characterization of workflow systems as used during discovery, production and distribution of science. Vidger et al. [96] discussed their experience of automating a workflow management system and described its benefits for supporting their software such as ease of use, management of their activities and integration of their software tools.

Problem solving environments (PSE) was discussed by Houstis et al. [109]. They defined PSE as "a computer system that provides all the computational facilities necessary to solve a target class of problems" to address some difficulties of computational science such as the difficulty of physical simulation, high cost and time to develop the software, increase the availability of SES components and reliability of simulations.

Other observations

Beazley et al. [131] in their paper described their experience of using Python for developing a large-scale application for parallel processing systems. They identified several problems that could be addressed by using Python. The first problem occurred as their simulations usually generate huge amount of data which needed to be analyzed. To perform such analysis on user's workstation or to buy everyone their own personal desktop supercomputer didn't seem feasible. The second challenge emerged as they were required to make constant changes to the application code (written in C), which they found to be very tedious with low flexibility.

Beazley in his paper [130] discussed developing an extensible compiler to automate the integration of compiled code with scripting language interpreters. Integrating compiled code with an interpreter is very common challenge when using scripting languages.

In the automotive industry, a huge amount of code is still written by hand [105]. There are some tools for generating code, though those tools are not efficient enough to produce optimal code. On the other hand "highly optimized code makes reuse and maintenance quite hard" as stated by Broy [105].

According to Tang's thesis [26], in terms of coding standards, it was concluded that industry is much more careful than academia with respect to implementation standards. 31%, 66% and 69% of the respondents reported the use of specific tools in code generating, program debugging and version control.

Arnold et al. [123] in their paper has described their proposed SCAI (Scientific Computing Application Infrastructure). This infrastructure provides the user (novice non-computer scientists) with an easy programming model and API as well as incorporation of different types of computational modules. Also it supports different granularities of computational modules. Module complexities are hidden while they are easily accessible. Scripting is supported to let the user combine modules. The infrastructure also support different languages and keeps the performance optimized.

Dall'Osso [110] in his papers discussed the advantages of using CASs (Computer Algebra Systems) in automatically generating programs. The purpose of these systems is to enable the people who are just familiar with the physics of the problem to write code without being involved with numerical algorithms. The CAS approach supports incremental development and thus the problem formulation can be improved after the correctness of the current version is verified. Also if a change occurs and the program needs to be updated, just the specifications from which the program is created needs to be updated.

3.6.2.4 RQ 2.4: What are the challenges and solutions in testing scientific software?

We had 11 papers presenting the issues related to testing SES, including one expert view, two case study, two surveys, one experiment and five concept implementations. Table 13 summarizes the information related to these papers.

Ref	Paper's main focus	Evidence type	Challenges	Solutions/Observations	Specific to SES	Context/ Domain
[78]	Dealing with risk	Survey	- Lack of test oracles - Complexity of functionality verification - Complexity of software validation		Yes	A mixture of engineering and scientific disciplines
[23]	Identification of the steps and tools in developing high-performance software	Case study			Yes	High performance computing
[33]	Improving SS development	Expert view			Yes	General
[97]	Modeling the input space for testing	Case study	Manual selection of enough test cases	Solution: a model to capture the dependencies among the input space for automated test generation	No	Multiphysics simulation

[76]	Proposing mutation sensitivity testing for testing SES	Concept impl.	Dealing with Tolerance problem		No	General
[25]	Presenting a methodology for development of the requirements for general purpose scientific computing software		Testing continuous values		No	General
[75]	Testing SS		Large number of required test cases	Observation: small number of well-chosen test cases may reveal a high percentage of code faults	No	General
[85]	Automated verification and validation technique for image segmentation		Verification and validation of medical image segmentation		No	Image segmentation software
[84]	Proposing the use of code mutation for testing SS		The difficulty of detecting silent faults (i.e. code faults, not scientific calculation inaccuracy)	Solution: mutation sensitivity testing (reducing error tolerance is much more effective than running more tests)	Yes	General
[50]	Surveying how scientists develop and use SS	Survey		Observation: separation of software bugs from model errors are not addressed yet by software testing community	No	General
[128]	Investigating errors in SS	Experiment		Observation: SES code is not as accurate as expected	Yes	Seismic data Processing

Table 13: Summary of the papers discussing testing issues

In the following sections more details on the challenges, solution and other observations concerning the testing in SES are presented.

Challenges

Testing scientific software in practice is a critical task because it is a two-fold problem. Firstly, "doing the right thing" or validation of what is really needed to be done is not an easy task, according to Segal and Morris [33], for the software engineer who is not as knowledgeable as the domain expert. Secondly, similar to any other software application, the need to test for "doing things right" or verification of the software remains another challenge in testing SES. This issue as a whole is not well addressed by scientists as is discussed by Sanders et al. [78]: "If the software's purpose shifts away from just showing the theory's viability, risk shifts to the implementation. At this point, testing must assess the implementation, not the theory. Most scientists miss this shift".

In the cases, where the entire purpose of developing the software is to solve a problem that does not currently have a solution, the validation of the end product is very complex, if not impossible, as stated by Carver et al. [23].

Segal and Morris [33] also stated that the lack of "test oracle (expected output)" is a main factor that makes testing SES difficult in many domains. Most of the time, valid data against which the output of the software can be compared does not exists and it is very hard to build a rigorous test oracle. This will cause a challenge called "tolerance problem" as reported by Hook and Kelly in [76], which is mainly the result of having uncertain oracles and other errors such as rounding error [83] caused by floating point representation.

As reported by Smith in [25], the fact that some of scientific applications use continuous values in input and calculation further adds to the complexity in the validation of these systems. This is since success in one test case does not imply success in another test case containing nearby values, since that nearby value may be a boundary value in the defined scope of the variable under test or a singular value which makes one equation undefined or cause a division by zero at some point.

As reported by Vilkomir et al. [97], in most simulation software such as multiphysics, the huge number of input values and parameters make the manual selection of sufficient test cases very complex.

Another major testing-related challenge, reported in [75] by Hook and Kelly, is the large number of test cases required when following any standard software testing technique described in the literature, e.g., category-partitioning, or code coverage-based testing.

Frounchi et al. [85] have discussed the challenge of verification and validation of medical image segmentation, which is usually performed manually by an expert. In this verification and validation process, if the result is not satisfactory, the segmentation algorithm needs to be revised and again the outcome should be evaluated by the expert in an iterative manner.

Possible solutions

Hook, in his thesis [84], proposed employing "mutation sensitivity testing" to resolve the challenge of detecting "silent faults" in scientific code, i.e. code faults, not scientific calculation inaccuracy. In this method, the pass/fail criterion (i.e., test oracle) is not based on the equality of the expected and actual outcomes (often, outputs). Rather, the pass/fail criterion is based on the mutation sensitivity of each test case. This way, the traditional mutation testing can be used as a tool for computational software testing.

To tackle the issue of manual selection of enough test cases, Vilkomir et al. [97] proposed a model which captures the dependencies among the input space and then test cases can be generated automatically from the model.

Other observations

Initial research results by Hook and Kelly [75] suggest that a small number of well-chosen test cases may reveal a high percentage of code faults in scientific software and allow scientists to increase their confidence.

More recently, there have been further developments in the area of testing SES. For example, a testing process model for scientific software by Hook and Kelly was presented in [75]. The model consists of three different levels of activities each of which address a main need in testing. The first level assesses whether the software can be used by scientists or engineers in order to get their work done. This level is called "scientific validation". The second level which is called "algorithm verification" which assesses the relevance and the strength of the methods and approaches used to solve the problem of scientists and engineers. The third level or the "code scrutinization" tries to detect code faults and other problems that occur while using computer languages.

Hannay et al. reported in [50] that scientific software testing raises issues that have not yet been addressed sufficiently by the software testing community. Issues such as separating software bugs from model errors and approximation errors or not having a certain test oracle available are the issues that can not be easily addresses just by referring to common software testing practices and approaches.

In [128] Hatton reported the details of two experiments conducted to measure the accuracy of SES code. The first experiment aimed at measure the consistency of millions lines of code written in C and Fortran. The second aimed at measuring the level of dynamic disagreement between different implementations of the same algorithms working with the same input data and the same parameters. As a result they found out that code is not as accurate as expected.

We also experienced the same challenges as reported in the literature. Our experience in our ongoing optimization software development project for oil pipelines' pump operation [132], again provides empirical evidence on the issues of testing SES. As part of the project, we are building an optimization algorithm and tool (details in [134]) which takes as input the pipelines information (e.g., topography, pump settings, etc.) and provides as output an optimal operational regime (configuration) for pump speeds which would deliver the contracted volume of oil product(s) while minimizing the dollar cost of electricity used to pump the product(s). The back-end of this optimization tool is developed using a commercial optimization solver, called LINDO [135], and a .Net-based front-end (GUI) is utilized.

Validating the outputs generated by this optimization algorithm and tool and whether they are actually optimal is not trivial. One option is the real-world log data however it is almost certain that real-world settings were not optimal. As another option, we are planning to develop another optimization tool based on other optimization techniques (e.g., genetic algorithms).

3.6.2.5 RQ 2.5: What are the challenges and solutions in maintenance stage of scientific software?

We found six papers mentioning the issue related to the maintenance of SES, including two surveys, two concept implementations, one experience report and one expert view. Table 14 summarizes the information presented in these papers.

Ref	Paper's main focus	Evidence type	Challenges	Solutions/Observations	Specific To SES	Context/ Domain
[50]	Surveying how scientists develop and use SS	Survey	Ignoring maintenance while developing	Observation: maintenance of SES is moderately important	No	General
[26]	Developing scientific and computing software			Observation: the lifetime of typical SES is long	No	General
[120]	Integrating a technological and design approach to support SS evolution	Concept implem.	Dealing with fast evolving domains	Solution: defining different iterations in software development	No	Biology
[99]	Introducing a tool to perform rapid component prototyping while maintaining robust software engineering practices		Addressing the issues of component glue code		No	High-performance scientific computing
[131]	Using python in SS development	Exper. rep.	Dealing with the situation where different users modified their own copy of the software		No	Large-scale physics application
[105]	Developing software for automotive industry	Expert view	Long-term maintenance is required		No	Automotive software development

Table 14: Summary of the papers discussing maintenance issues

In the following sections more details on the challenges, solution and other observations concerning the maintenance in SES are presented.

Challenges

One of the challenges of maintaining scientific and engineering code emerges from the fact that the focus in most of SES developments is primarily on developing working software in shortest possible time. This way, most of the software engineering practices, which came to existence to help manage the complexity of maintaining SES, is usually ignored as reported in the survey by Hannay et al. [50].

Allan et al. in their paper [99] discussed the difficulty of maintenance even when small amount of code is needed to be added to reusable components and libraries. Addressing the issue of component glue code and software build process is reported by the authors to be tedious and error-prone.

Beazley et al. [131] in their paper described the challenge of development and maintenance of their software. They were a small group of people using the application, thus "different users had their own private copies of the software that had been modified in some manner" and that led to a "maintenance nightmare that made it almost impossible to update the software or apply bug-fixes in a consistent manner".

Long term maintenance is mentioned as a challenge in software engineering for automotive industry as "the cars are supposed to be in operation over more than two or three decades" [105].

Possible solutions

In general, the issues of software evolution are addressed by defining iterations of software development or maintenance cycles as suggested by Letondal and Zdun in

[120], but this is not sufficient to resolve the issue of having fast evolving domains, which will result in the need for fast evolving software.

Other observations

According to the survey by Tang [26], the lifetime of typical SES is long as just 4% of the software were reported to have lifetime shorter than one year. 70% of SES software is planned to be used for more than 6 years and 22% has a lifetime of more than 20 years. Also based on a recent survey by Hanney et al. [50], the importance of scientific software maintenance is ranked moderately important by the scientists who participated in the survey.

SES maintenance is the phase which has not gained much attention from researchers (the number of publications focusing on this phase is noticeably low), regardless of its undeniable importance.

3.6.2.6 RQ 2.6: What are the challenges and solutions in cooperation and human-related factors of scientific software projects?

In our pool of papers, seven papers reported the issues regarding the cooperation and human-related factors in SES development including one experience report, two filed studies, one concept implementation, one expert view and one survey, as summarized in Table 15. In the following sections more details on the challenges, solution and other observations concerning the cooperation and human-related issues in SES development are presented.

Challenges

SES developers come from different disciplines in science and engineering such as physics, biology applied math, civil engineering and computer science. The "large variability in specialized backgrounds makes collaborative software development difficult" as stated by Bartlett [87].

Ref	Paper's main focus	Evidence type	Challenges	Solutions/ Observations	Specific to SES	Context/ Domain
[87]	Proposing different integration strategies for computational science and engineering software	Experience report	Collaboration among various disciplines is problematic		No	General
[81]	Culture and cooperation problems in SS development	Field study	Communication issues between management and developers		No	Biology
[80]	Challenges of software engineers		User engagement in design and development		No	Software engineers developing software for space scientists and biologists
[114]	Managing individualist programmer	Con. imp.	Managing programmers who prefer to develop in isolation	Solution: Following management policies	No	General
[3]	Identification of the gap between software engineers and scientists	Expert view		Observation: developers can be classified under 3 groups: industrial developers, scientific and engineering researchers and students	Yes	General
[5]	Identification of problematic issues in scientific computing			Observation: developers need to be trained to successfully employ SE methodologies	No	General
[50]	Surveying how scientists develop and use SS	Survey		Observation: the lack of formal training is very common	No	General

Table 15: Summary of the papers discussing cooperation and human-related issues

In a field study [81], Segal has identified communication issues between the management staff and developers. These issues become problematic particularly because of the interference of the managers in technical issues. In one industrial setting, Segal [81] reported that the management board was expected to negotiate, prioritize and make final decisions about the incomplete and ambiguous requirements, but they failed to do so properly. The developers often interpret such a failure as interference in technical issues without any noticeable success in resolving them. On the other hand, the developers often fail to collaborate with the lab scientists because of not being managed properly from a higher practical level.

The challenge of user engagement in design and development of SES considering the fact that the majority of scientific software users are scientists is important to tackle stated by Segal in [80]. The reason is quite obvious: the lack of software engineering knowledge about the problem and the problem domain.

Possible solutions

Hovendon et al. [114] recommended that in terms of managing and directing the project to the right path for building high quality products, the management policies are of great importance specially to harmonize the individual programmers toward the same goal.

Other observations

Kelly in her paper [3] divided the scientific developers into three major groups: (1) the industrial developers who are engaged with applications related to their domain of expertise, (2) scientific researchers and (3) students who will be identified as one of the two other mentioned groups based on their choice of career. Each of these three groups have their own approach toward application development, but all of them need to be properly aware and academically trained in order for employing software engineering methodologies to experience successful development practices as Wilson stated in [5].

According to the survey by Erskine et al. [50], the lack of formal training is very common and the usual training that scientists might experience is offered by the computer science department, which is mostly domain-independent and general as reported by Kelly in [3].

3.6.3 *RQ 3- What are the best practices in SES development?*

Because of being very dependent to the specific characteristics of a particular discipline for which the software is being developed, it is very difficult to propose a unique and universal framework or methodology which can well be applicable to SES development in all domains. However we have found several practices suggested by researchers and experts who could efficiently develop quality products. These practices are grouped and tabulated in Table 16.

	Best Practices	**Context/ Domain**	**Ref**
Requirements	Having a user and system requirement document to specify the functional, performance and the interface requirements of the software	General	[51]
	Responding to immediate emerging requirements and needs rather than building a complete solution	Scientific workflow management system development	[90]
	Determining the schedules and resource levels based on requirements	Large scale multi-physics computational simulations	[129]
Design	Having software design documents	General	[51]
	Using design patterns	Plasma physics	[118]
	Using component-based software architecture	Quantom chemistry - High-performance and high-end	[98, 124-127]

		scientific computing -	
	- Designing in a way which fits in the requirements of the scientists - Designing to support extensibility and customization - Designing to meet local needs while making the product easy to extend to cover more general needs	Scientific workflow management system development	[90]
	Designing the project upfront	Biology	[103]
Implementation	- Separation of the scientific calculation code from the user interface code and the data - Writing simple code - Having the code reviewed by other scientists	General	[74]
	- Pair programming - Creating source-centric documents	General	[88]
	Building core capabilities promptly	Scientific workflow management system development	[90]
Testing	Testing SES validity	General - Weather forcasting	[29, 74]
	Writing tests first and running them often (test-driven development)	General	[88]
	Having a test plan on the development of the testing strategy and test case generation	General	[51]
	Developing and executing a verification and validation program	Large scale multi-physics computational simulations	[129]
Development Process	Being organized in different stages and activities	Large scale multi-physics computational simulations - General	[74, 129]
	- Performing continuous process improvement - Managing the repositories - Using checklists for repeated activities	General	[88]
	Using configuration management tools	General - Weather forcasting	[29, 88]
	Using a formal release plan	General	[88]
	Having a management plan and applying project management techniques	General - Biology	[51, 103]
	Having quality control and assurance plan	General - Biology	[51, 103]
	Listening to customer	Weather forcasting	[29]
	Documenting the program and the key issues	Biology	[103]
	- Performing scheduling and estimation of	Large scale multi-	[129]

	resources based on code development experience - Identifying the risks	physics computational simulations	
Communication and Human Aspects	- Using issue-tracking software - Communicating by mailing lists	General	[88]
	- Having highly competent staff - Investing in people with training and support	Large scale multi-physics computational simulations	[129]
Deployment and Maintenance	- Having the system documentation - Preparing the user manual and installation guide - Running a web site to provide the users main contact points for bug reporting and release developments - Having a maintenance guide to manage bug reports, perform regression testing and redistribute the system	General	[51]
	Maintenance with customer focus	Large scale multi-physics computational simulations	[129]

Table 16: Best practices in SES development

3.7 Discussions on threats to validity of the results

There are always some sources of threats to the validity of a review, which result in the inaccuracy of the results. One primary source of inaccuracy, which is called threat to internal validity, is imprecise data extraction. To prevent this threat, we did our best to define our research questions as detailed as possible by including sufficient sub-questions to make sure that we address those questions precisely and with the exact related piece of information extracted from the primary studies. Also, we have specified the type of information (evidence) which is needed to address the review questions for each of the questions. By defining this framework, not only we can avoid biased judgments, but we can also discuss disagreements in depth with respect to the details which are extracted from the publications. This approach also prevents the threats to the construct validity (observation validity) of the results, which occurs when the observation method does not exactly capture what it requires to observe [136].

Threats to the external validity are conditions that limit the generalizability of the results. In this work we presented possible solutions to particular SES development challenges besides the best practices as applicable to certain types of the software systems. Our primary studies include publications covering a variety of domains and different types of software systems, yet certain conditions as applied to some software systems might occur, which are not considered in the publications, while proposing the solutions and practices to address the challenges. This type of inaccuracy, as well as another type of construct validity, which is called intentional validity (does the constructs we choose adequately represent what we intend to study? [136]) occurs when the repository of the publications is not complete and it does not contain all the relevant publications. We tried to decrease the possibility of this risk by searching through all the famous electrical resources and publishers by a comprehensive and precise search string which well represents the topic of this review. As we mostly select top journal papers and conference proceedings it is possible that we have missed certain relevant information when only presented in theses and technical reports. Also as we searched for the publications based on their title, when the title of the primary study does not match with our search key words, the article can not be found. As mentioned in the corresponding section, we defined our set of search key words in a way to cover all relevant titles.

3.8 **Chapter Summary**

In this section we presented our SLR on the role of software engineering in the development of SES. Developing SES is different from conventional software development practices mostly because its primary aim is to help the scientists and engineers better understand, analyze and resolve their domain issues and thus is highly tied with the knowledge and expertise of scientists as the real owners of the software.

For this review we designed a set of important research questions mostly on the challenges of SES development, extracted the relevant info from primary studies and then presented the potential solutions and other observations found in the literature. Best practices as applicable to different problem domains and various projects were also tabulated and summarised for practitioners.

The next chapter will provide basic information about the case study of developing engineering software for optimization of pipeline operation. The case study aims at providing evidence on the challenges and solutions for the development of engineering software.

Chapter Four: **Overview of the Oil Pipeline Operation Optimization Software Development Case Study**

In this chapter the overview of the case study we undertook, which was aimed at developing engineering software for the energy industry is provided. We start by introducing the main project and the team members in Section 4.1. In Section 4.2, the case study goals and research questions are presented and discussed, followed by the main domain terminology in Section 4.3, used to communicate with the domain experts. A brief description of the optimization problem is offered in Section 4.4 and overview of the pipeline under study is presented in Section 4.5.

4.1 **Project and Team Members**

The amount of energy required to operate oil distribution systems is huge. This energy is in the form of either electrical or fossil fuels, and is an enormous portion of the total expenses in transportation and distribution companies. The reduction of this energy is valuable in the sense that it saves a great amount of money for the companies as well as preserving the environmental cleanness by burning less fossil fuel. The energy reduction is the result of optimal operation of the oil distribution system. To achieve such optimal operation, the distribution system needs to be modeled mathematically and then that model can be optimized using a proper optimization method.

This optimization problem as a part of developing intelligent software solutions for energy industry along with the opportunity of working with industrial partners motivated our research group to develop an engineering software application which provides the oil distribution system operators with optimal operation settings as well as the decision support system to assist them in making proper choices. The system was planned to be developed for an energy transportation and service provider company located in Calgary named Pembina [137]. The team members collaborating with each other in this project include a principle investigator, two M.Sc students, a domain expert and a technical consultant. These roles along with the corresponding expertise are tabulated in Table 17.

Role	Description
Principle investigator	Software engineer and optimization expert (the author's advisor)
M. Sc. student	Software engineer (the author)
Master student	Optimization problem modeling and formulation, with partial domain expertise
Domain expert	Main correspondent in the company, pipeline operation expert
Technical consultant /Post-Doc. Fellow	Optimization and pipeline operation expert

Table 17: Team member roles and their expertise

4.2 **Case Study Research Process**

In this section the process of conducting the case study is described based on the guidelines provided in [138]. We went through three steps for this study: (1) design of the case study, (2) collection of the evidence, throughout the case study and (3) reporting the case study findings. Each of these steps will be discussed in the upcoming sub-sections.

4.2.1 ***Case study design***

In the beginning of the case study the plan for conducting the case study should be designed. Certain elements are required to be defined in the plan [138], such as objective of the case study, the case which is planned to be studied and research questions of the study. These elements will be discussed in the following sub-sections.

4.2.1.1 Objective of the Case Study

Objective of the case study describes what we expect to achieve out of the study. This case study, as a part of the major project, which was introduced in the Section 4.1, aims at developing a software system to provide the optimal operation regime and the decision support for the user by visualising the optimal pipeline parameters with different pipeline operation settings. These features are offered by the software we designed and developed in collaboration with another master student in our research group, who provided us with the optimization formulation module. The interested reader in the details of the optimization algorithm and pipeline hydraulics and operational formulations can refer to [139]. The details of the software we designed and developed to embed the optimization engine, which provides the user with some decision support features are given in Chapter 5, 6 and 7. The objective of the case study can be summarized as "assessment of challenges and lessons learnt in the development of an oil pipeline operation optimization and decision support software and its overlap with the SLR findings".

4.2.1.2 The Case

The case, describes what is planned to be investigated under the study. This is referred to as any "contemporary phenomenon in its real-life context" in [138]. Thus, here we consider "the development of the optimization software and decision support system for oil industry" as our case.

4.2.1.3 Research Questions

In parallel with developing the optimization software, we also defined 2 research questions to be investigated in the case study. The research questions are inspired by the findings of the SLR in different stages of the development and we summarize them as follows:

- Case Study RQ1: What are the particular challenges of developing oil pipeline operation optimization software?
- Case Study RQ2: How the challenges of developing oil pipeline operation optimization software can be addressed?

Based on the findings of the SLR, the research questions can be further refined to a set of hypotheses as summarized below.

Hypotheses based on the Cases Study RQ1:

- H1.1: Gaining domain expertise is time-consuming and difficult for software engineer, compared to learning the basics of typical non-scientific/non-engineering domains.
- H1.2: The requirements cannot be decided in early stages of the development as they evolve throughout the process.
- H1.3: Test oracles are uncertain, as often there is no prior solution for the problem at hand.

Hypotheses based on the Cases Study RQ2:

- H2.1: Regular meetings with domain experts are a beneficial practice for validation of the requirements.
- H2.2: Adopting iterative approach fits the "evolving and emerging requirement" nature of engineering software.

- H2.3: Adopting OO methodology, can pave the way for using design and architectural patterns besides giving a better management over data and functions.
- H2.4: The challenge of having uncertain oracles can be addressed by employing another independent method for solving the engineering problem, so that the similarities between the results achieved from two methods can be investigated to add more to the validity of the solutions.

4.2.1.4 Data Collection Method

In order to address the research questions described before, we choose to collect the data based on observations in different stages of developing the software, which is one of the qualitative methods of data collection in software engineering [140]. According to [138], the benefit of observations is mentioned to be the possibility of providing a deep understanding of the phenomenon under study.

We followed two approaches in our observations for data collection [138]: (1) "think aloud" approach, where the subjects are repeatedly reminded to think aloud by asking questions such as "What is your strategy?" or "What are you thinking to?". Subjects in our case are software engineer and domain experts. (2) Observation in meetings is another approach, where the observation data is generated during the meetings when participants interact with each other.

4.2.2 *Collection of the Evidence*

According to the data collection method described in previous section, we observed and recorded our data in different stages of the development. The evidence generated through meetings with our industrial partner for learning the domain basics and requirement elicitation, as well as our own observations of the experience of designing, implementing and testing of the software, was collected. Also the thoughts of another member of our research group, who was responsible for the development of the optimization module, was frequently gathered, using "think aloud" approach.

4.2.3 ***Reporting***

In Section 5.8, 6.3 and 7.4, for each stage of application development, we have summarized our observations regarding the certain challenges of developing this engineering software besides presenting the solutions and observations we had during the life cycle of the application.

4.3 **Basic Domain Terminology**

The objective of the project is to develop software for optimizing pump unit selection which provides the operator with the optimal operation strategy for all the pump stations in the pipeline distribution system while maintaining the desired delivery schedule [139].

In this section we present basic descriptions of the concepts we had in our target domain. This domain terminology is required for proper understanding of the problem domain while communicating with domain experts and reading technical documents. Also it assisted us in the general understanding of the problem and gathering the required information for this study.

4.3.1 ***Pipeline Systems***

A pipeline network is a system of pipe segments, pumps, values and other related instruments which are used for delivering fluid or gas products from source points to designated target points. A snapshot taken from the Alaska pipeline is shown in Figure 12.

Figure 12: Snapshot taken from Alaska pipeline (taken from [141])

Pipelines in this problem are used to receive oil products and deliver it to several terminals at a predefined schedule that includes target volumes over given time periods. Typically, storage is available at the initial port and several intermediate locations. The pipeline scheduler is provided with a set of contractual constraints that define the target deliveries at various locations. The system is supposed to generate efficient configuration and operating regimes for the system based on problem objective, which is the reduction of power expenses.

4.3.2 *Pump*

Pipeline systems usually are spread along very large distances. As an example we can refer to the length of the oil pipelines of the largest operator in North America (Enbridge Inc.) which is over 5,000 km [142]. Thus products may require travelling a very long distance to reach their certain target point. Factors such as friction between the product and the pipeline internal surface and differences in altitude result in loss of primary pressure which was used to pump the products in the pipeline at source points.

This pressure should not fall under a certain threshold, or the product flow rate in the pipeline will be corrupted. Pumps are used along the pipeline in order to keep the products moving in pipeline with a reasonable flow rate, in order to meet the contractual constraints in the right time.

Pumps in this problem are either fixed speed or variable speed centrifugal pumps. In variable speed pumps the operating speed can be adjusted as required, while in case of fixed speed pumps, the pump operated with a certain constant speed.

Pump operating characteristics are typically demonstrated by pump curves which are provided by the pump manufacturer. These curves depict the relationships between the following parameters [143]:

- Pressure produced by the pump which is called head pressure. Head is measured by the height of liquid stub. It is basically the difference between pump suction and pump discharge pressure,

- Flow rate is the amount of liquid passed through the pump in the certain time unit,

- Speed by which the pump turbine is rotating (rounds per minute),

- Pump Efficiency is the measure of how efficient the input electrical energy is transformed to output pressure,

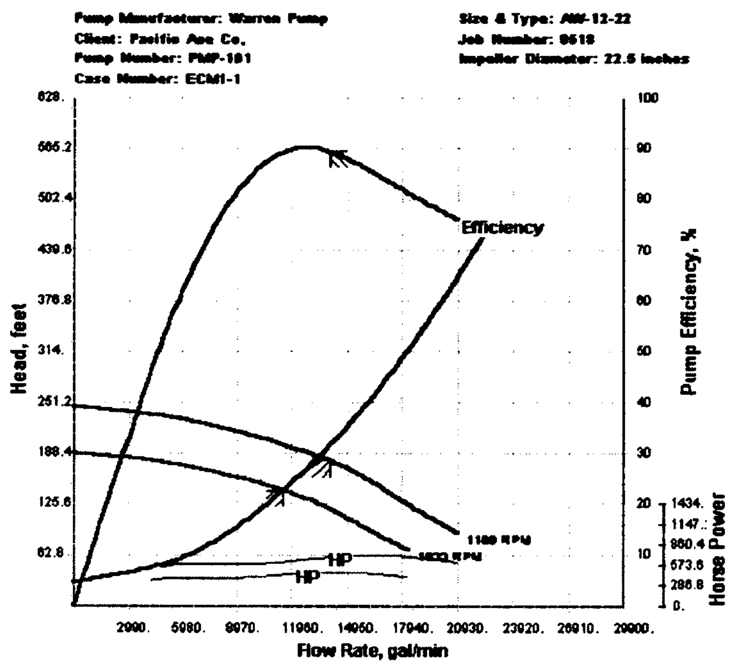


Figure 13: Sample pump curve, head vs. flow rate and efficiency vs. flow rate (taken from [143])

A sample pump curve is shown in Figure 13. It is worth noting that the pump characteristics are subject to change after being used for a long time and need to be updated.

4.3.3 ***Pump Station***

Pump Station is a location where one or more pumps are placed. Pumps typically are connected serially or in a parallel fashion. There exist cases where the connection between pumps in the system is even more complex with some of the pumps connected parallel and others serially. Every pump station can have multiple inlets and outlets. A snapshot taken from a pump station which has four pumps is shown in Figure 14.

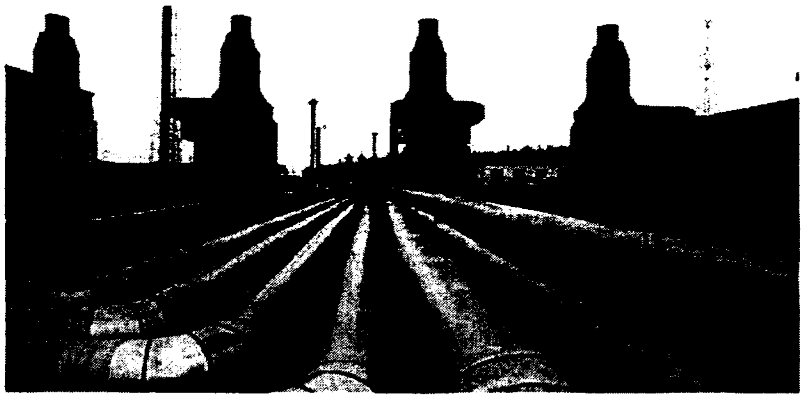


Figure 14: A pump station with four pumps (photo by Sergei Grits [144])

4.3.4 *Control Valves*

In order to balance the pressure or flow rates in specific points of the pipeline, control valves are used to maintain the desired condition at those points. Valves are mostly placed in the positions where the pressure needs to be reduced. A snapshot of two control valves, taken from an oil pipeline is shown in Figure 15.

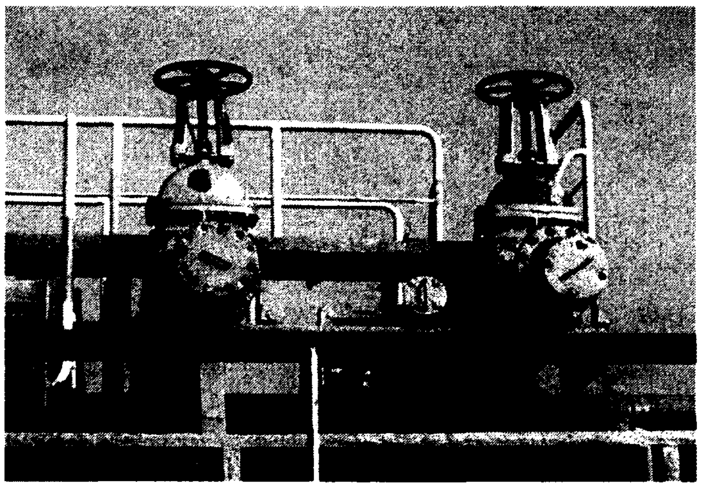


Figure 15: Oil pipeline control valves (adapted from [145])

4.3.5 ***Power Contract and Power Rate***

Power Suppliers are companies providing electrical or other kind of power for the pipeline system. Every Power Supplier supplies electricity for one or more pump stations, while one pump station is usually supported by one power supplier. Power cost is negotiated with every power supplier and the final agreement is signed in power contracts for a certain period of time.

Power contracts consist of various cost rates and their thresholds that are used to calculate the power consumption cost for running each pump. When a threshold for the first cost rate is reached, the second cost rate is applied until the second threshold is reached and so on. Additionally, power contracts have their start and end dates which define the period in which the contract is valid. Generally the power contracts can be more complex by having different cost rates within a day, week or month.

In our optimization problem, we considered one threshold for electricity cost rates. Two sample electricity cost rates of this type are shown in Figure 16.

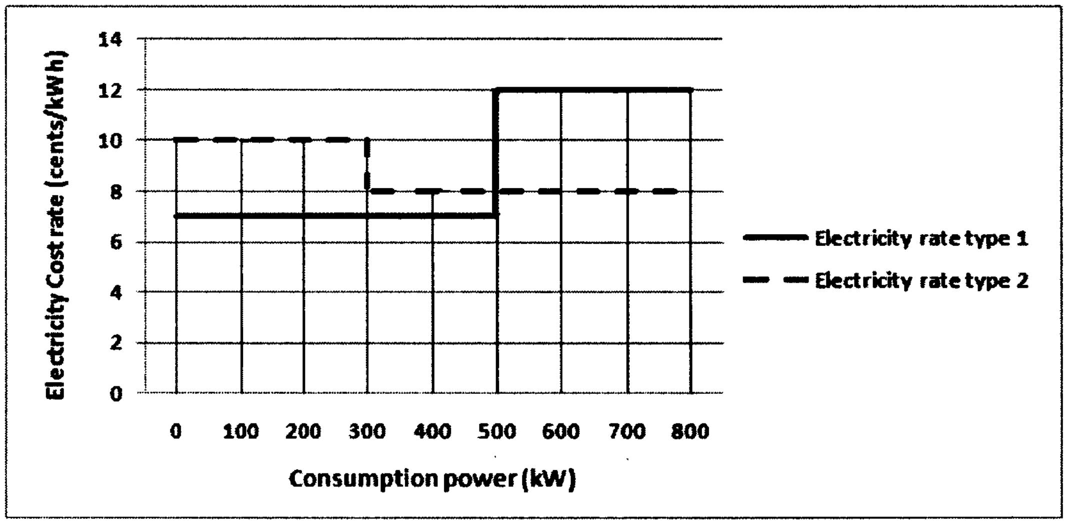


Figure 16: Two sample types of electricity cost rates

4.4 Overview of the Optimization Module

The aim of optimization in this project was to find the optimal operation setting of the oil pipeline. This is known to be a complicated problem because of having huge number of integer variables and the other hydraulics non-linearities involved. On the other hand, as the technique was expected to be applied to a real working system, it was required to be a reliable and efficient solution. By reliable we mean the solution should be able to find the closest optimal values to the global optimum and by efficient we mean the execution time of the algorithm should be reasonable to operators, to make the system usable for them. Main decision variables in this problem formulation include:

- the status of the pumps (which means if they are in operation or not),
- the system flow rate,
- added pressure by each pump unit in each pump station,
- power cost rates and thresholds for each station,

In order to address the specific requirements of the system, as mentioned briefly above, Mixed-Integer Linear Programming (MILP) was selected to be used as optimization method in this problem. The power of MILP lies in the fact that it guarantees the convergence to the global optimum in finite number of iterations while providing a flexible and accurate modeling framework [146].

The optimization module includes a commercial optimization solver, and the optimization formulation file. As mentioned before, the optimization formulation file was designed and developed by another master student working in our research group. He developed the formulation file by first defining a formulation framework in which the decision variables, sets and fixed parameters were identified after gaining the required domain expertise. This domain expertise was acquired by studying the pipeline networks theoretical basics and through meetings with our industrial partner. Then the relationships among the decision variables, fixed parameters and sets were identified and the constraints were taken into account in order to obtain the formulation for the entire pipeline.

He evaluated the meetings with the industrial partner helpful in understanding the case study system basics. Collaborating with domain experts also was beneficial for the correctness of the formulation file and the verification of the optimization results.

The main challenge he faced was the confidentiality of the pipeline network data and software usage which resulted in some extra works to hide that information. Also the experts were sometimes tardy in responding back to the requests which resulted in significant delays.

Interested reader can refer to his thesis [139] for more details on the decision variables, objective function, optimization formulation logic and the optimization technique used in this problem.

4.5 **Pembina Pipeline**

In this research project [132], we made an agreement with Pembina engineers to just considered the portion of Pembina pipeline starting from S1 station in Alberta and ending in Kamloops in British Colombia. This portion in total includes six pump stations, one of which is not currently in operation. Figure 17 shows the whole pipeline network under the operation of Pembina.

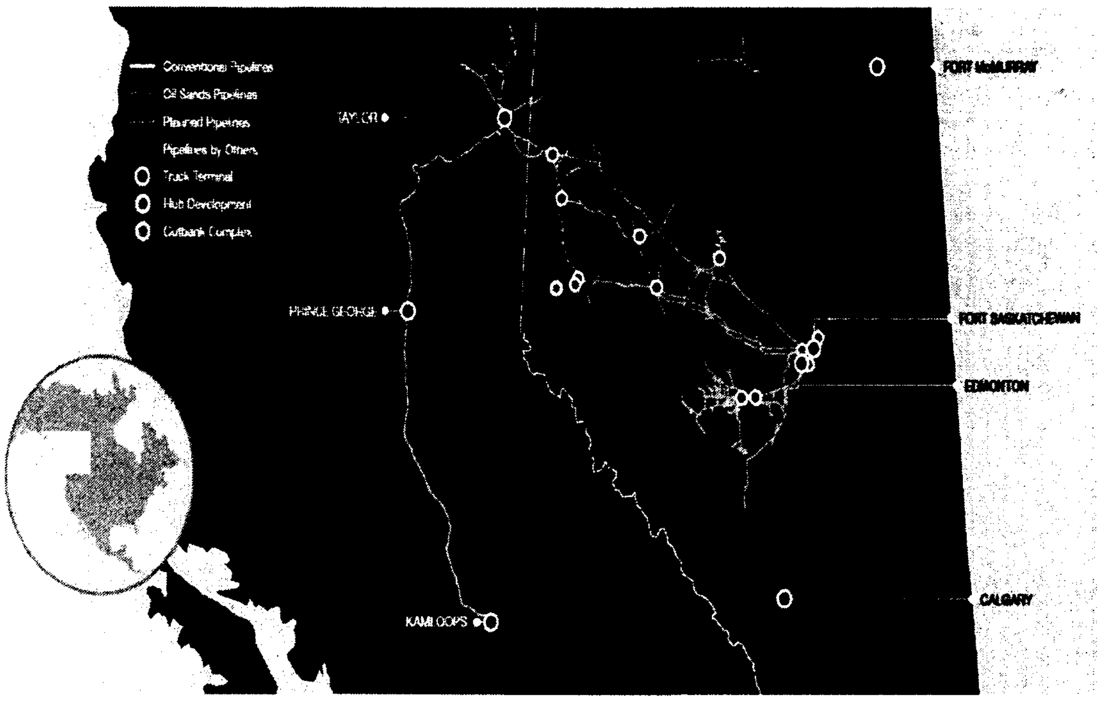


Figure 17: Geographical spread of Pembina pipeline [137]

The length of the pipeline in this study reaches to around eight hundred kilometres with two source nodes and two delivery points as shown in Figure 18.

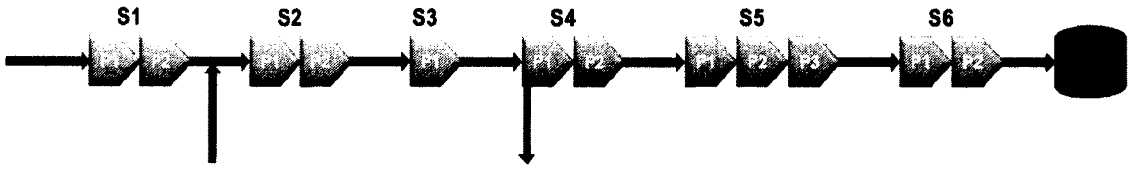


Figure 18: Schematic view of Pembina pipeline

To control the flow rate and pressure of certain points, around ninety valves are placed on the pipeline, twenty three of which are being used and monitored frequently. The name list, type and number of the pumps in each pump station are shown in Table 18.

In order to sketch the pipeline elements on Google Earth, we needed the geographical profile of the pipeline portion under study. This was achieved from the map of the pipeline provided by Pembina. For the sake of the confidentiality of the data provided by the company, we do not disclose the geographical profile of the pipeline elements in this thesis.

Pump station	Type	# of pumps
S1	Variable speed	2
S2	Variable speed	2
S3	Fixed speed	1
S4	Fixed speed	2
S5	Variable speed	3
S6	Fixed speed	2

Table 18: Pump stations in Pembina pipeline covered in this project

4.6 Chapter Summary

This chapter presented the information related to the major research project in our research group on the development of software solutions for optimizing the pipeline operation. The case study described in this thesis is a part of that major project. The goal of this study is the development of the engineering software application to provide the optimal operation regime for the operators as well as the possibility to visually inspect the pipeline important variables such as total power consumption, power cost for each of the stations and the pump speeds in variable speed pump stations.

Main domain terminology used to communicate with domain experts and to understand the system are elaborated briefly. Elements such as valves, pumps, and pump stations are the main components of each pipeline system which are represented in the optimization problem formulation by certain variables. The optimization problem, which is expected to be solved by the commercial solver embedded in the application, is also briefly described in this chapter. Our industrial partner and the pipeline network under study were presented.

Next chapter will discuss the requirements of the case study, besides elaborating the analysis and design of the application.

Chapter Five: **Requirement Specification, Analysis and Design**

This chapter summarizes our engineering software requirements and provides some of the important detailed documents used in the analysis and design of the system. In Section 5.1 the requirements of the system are introduced. Section 5.2 briefly presents the object-oriented design methodology employed. Actors, external systems and storage are presented in Section 5.3. Section 5.4 presents the use-case diagram, followed by activity diagrams in Section 5.5, architecture in Section 5.6 and class diagram in Section 5.7. The discussion of the experiences in this step of the development regarding the case study research questions is presented in Section 5.8, which concludes this Chapter.

5.1 **System requirements**

In this section we briefly present the requirements of the system.

5.1.1 ***Functional Requirements***

The optimization software is expected to provide several functionalities for the users who are mostly pipeline operators. First, the user should be able to login to the system. The user should be able to launch Google earth, load his/her target pipeline in Google earth and easily navigate and browse pipeline, pump stations, valves and other belongings of the pipeline and read the information attached to these objects as specification boxes easily.

The user should also be able to run the optimization engine, open a new optimization formulation file or modify the existing file. The optimized values after running the optimization engine on the target optimization file should then create comparison charts for each station and also for the whole pipeline in order to give the user the possibility of visually studying the results and comparing the suggested optimal values with the available historical data of the pipeline system. By optimization engine or optimization solver in this project we mean Lindo [147].

5.1.2 *Non-functional Requirements*

As we mentioned before ,the system is being developed for our industrial partner, Pembina [137], which is an energy transportation and service provider company located in Calgary. Obviously the system is expected to be user friendly and easily learnable. The system is expected to be secure as it contains the operation data related to the company.

The system should support the modifications (modifiability), replacements and extensions (extensibility) that might occur to Pembina pipeline, such as pump replacements, power contract renewal with updated power cost rates. It also should be easily customized (customizability) for other companies, in the way that they can load their desired pipeline and get the corresponding visual and optimal data for that pipeline from the system. As a result, the code should be easily maintained (maintainability).

Testability should also be considered during the system design, as we require performing automated testing on the system to assure its correct functionality.

5.2 **Object-Oriented Analysis and Design**

We have adopted object oriented analysis and design methodology to benefit from its advantages, such as improved maintainability and modifiability as mentioned in Chapter 3. There we discussed in more details about the challenges faced in the design of SES and object oriented methodology, which was mentioned as a potential solution to address the design challenges as well as leading to a robust, easy to maintain system. Compared to procedural design which is the most common practice among non-software engineers; object oriented methodology is a proper technique to manage the data and functionalities in complex SES.

5.3 Actors, External Systems and Storage

In the following sub sections, we discuss the elements that interact with the system. The list of these elements, which includes actor, external systems and storage along with their descriptions are summarized in the Table 19.

Name	Role	Short Description
Operator	Actor	The user who works with the application
Optimization Engine	External system	The Lindo optimization engine (solver) which runs the optimization file and returns the optimized values
Google Earth	External system	GoogleEarth application
MS Excel		Microsoft Excel
Text File	Storage	File containing extracted target optimal values from Lindo output file
XML file	Storage	File containing the pipeline specifications
Optimization formulation file	Storage	File containing the pipeline network formulation

Table 19: System list of actors and short definitions

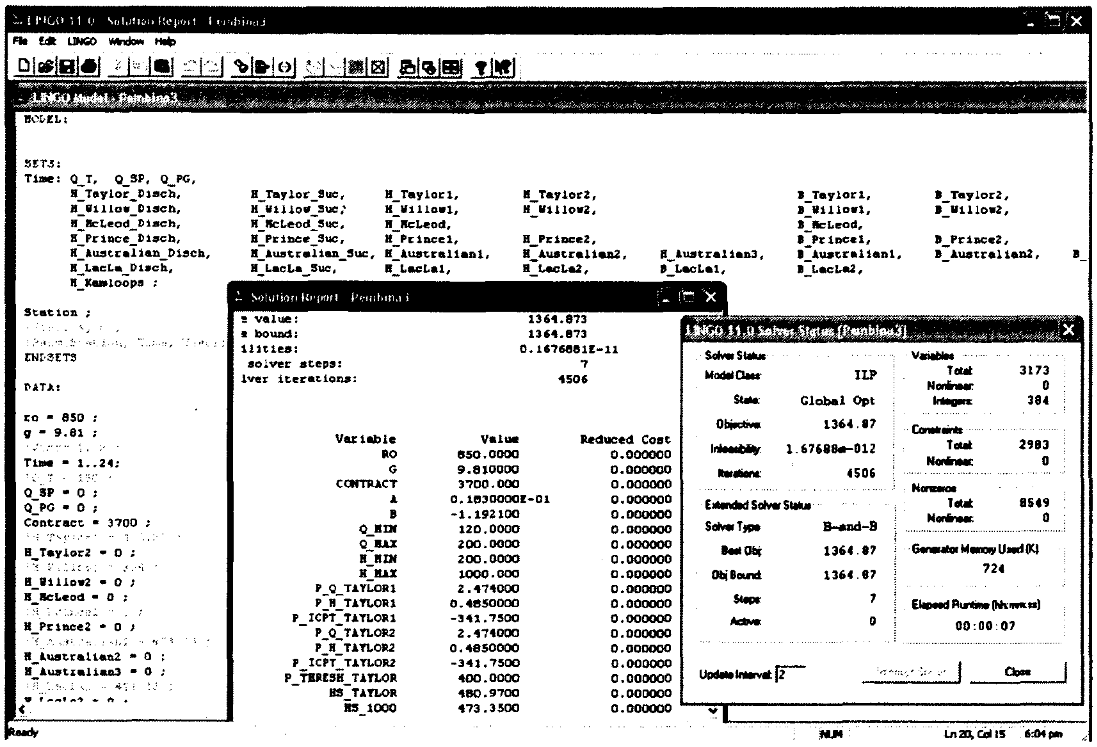
5.3.1 *Operator*

As mentioned above pipeline operators are the main users of the system. They start by logging in to the system and then browsing the network, defining/modifying system information, viewing system logs, viewing station's graphical views and other provided information on Google Earth interface, run optimization engine and have the possibility to view different optimization charts.

5.3.2 *Optimization Engine*

In order to optimize the pipeline operation cost, we used a commercial optimization solver named Lindo [147] which is responsible for getting the latest pipeline network information, formulated in a file, from the system and find optimal values based on which the pipeline operation will be optimized.

Lindo (independent from our application) provides the user with an environment, called LINGO, which integrates an editor for the optimization problem formulation and menu options for parameter setting and running the solver. A snapshot of LINGO is shown in Figure 19. In the figure, the LINGO environment, a sample model, which is in general the formulation of the problem to be optimized, the report produced after running the solver and the LINGO solver status are shown.


LINGO 11.0 - Solution Report - Pembina3
File Edit LINGO Window Help
LINGO Model - Pembina3
Solution Report - Pembina3
LINGO 11.0 Solver Status [Pembina3]
Close
Ready
NUM
Ln 20, Col 15 6:04 pm

```
MODEL:


SETS:
Time: Q_T,  Q_SP, Q_PG,
      H_Taylor_Disch,        H_Taylor_Suc,      H_Taylor1,        H_Taylor2,                          B_Taylor1,        B_Taylor2,
      H_Willow_Disch,        H_Willow_Suc,      H_Willow1,        H_Willow2,                          B_Willow1,        B_Willow2,
      H_McLeod_Disch,        H_McLeod_Suc,      H_McLeod,                                             B_McLeod,
      H_Prince_Disch,        H_Prince_Suc,      H_Prince1,        H_Prince2,                          B_Prince1,        B_Prince2,
      H_Australian_Disch,    H_Australian_Suc,  H_Australian1,    H_Australian2,    H_Australian3,    B_Australian1,    B_Australian2,    B_
      H_LacLa_Disch,         H_LacLa_Suc,       H_LacLa1,         H_LacLa2,         B_LacLa1,         B_LacLa2,
      H_Kamloops :

Station ;
[illegible]
ENDSETS

DATA:

ro = 850 ;
g = 9.81 ;
[illegible]
Time = 1..24;
[illegible]
Q_SP = 0 ;
Q_PG = 0 ;
Contract = 3700 ;
[illegible]
H_Taylor2 = 0 ;
[illegible]
H_Willow2 = 0 ;
H_McLeod = 0 ;
[illegible]
H_Prince2 = 0 ;
[illegible]
H_Australian2 = 0 ;
H_Australian3 = 0 ;
[illegible]
```

```
e value:                                  1364.873
e bound:                                  1364.873
ilities:                              0.1676881E-11
 solver steps:                                   7
lver iterations:                              4506


                 Variable           Value        Reduced Cost
                       RO        850.0000            0.000000
                        G        9.810000            0.000000
                 CONTRACT        3700.000            0.000000
                        A       0.1830000E-01        0.000000
                        B       -1.192100            0.000000
                    Q_MIN        120.0000            0.000000
                    Q_MAX        200.0000            0.000000
                    H_MIN        200.0000            0.000000
                    H_MAX        1000.000            0.000000
              P_Q_TAYLOR1        2.474000            0.000000
              P_H_TAYLOR1       0.4850000            0.000000
           P_ICPT_TAYLOR1       -341.7500            0.000000
              P_Q_TAYLOR2        2.474000            0.000000
              P_H_TAYLOR2       0.4850000            0.000000
           P_ICPT_TAYLOR2       -341.7500            0.000000
         P_THRESH_TAYLOR         400.0000            0.000000
               HS_TAYLOR         480.9700            0.000000
                  HS_1000        473.3500            0.000000
```

```
Solver Status                          Variables
  Model Class:            ILP             Total:        3173
        State:     Global Opt             Nonlinear:       0
    Objective:        1364.87             Integers:      384
  Infeasibility:  1.67688e-012         Constraints
   Iterations:           4506             Total:        2983
                                          Nonlinear:       0
Extended Solver Status                 Nonzeros
  Solver Type:        B-and-B             Total:        8549
     Best Obj:        1364.87             Nonlinear:       0
    Obj Bound:        1364.87          Generator Memory Used (K)
        Steps:              7                  724
       Active:              0          Elapsed Runtime (hh:mm:ss)
                                               00:00:07

Update Interval: 2
```

Figure 19: LINGO environment showing the optimization problem formulation, optimal solutions and the solver status

Our application is expected to call the solver which takes the formulation file saved in .lg4 format (the file format for Lindo models) and optimized the target objective function using MILP (Mixed Integer Linear Programming). The resulting optimal values are then extracted from the generated output file and saved in text files to be used later for chart creation.

5.3.3 *Google Earth*

The system should be able to interact with Google Earth in order to provide the user with the possibility of browsing different valves and stations graphically. A snapshot of Google Earth application, demonstrating Pembina pipeline is shown in Figure 20.

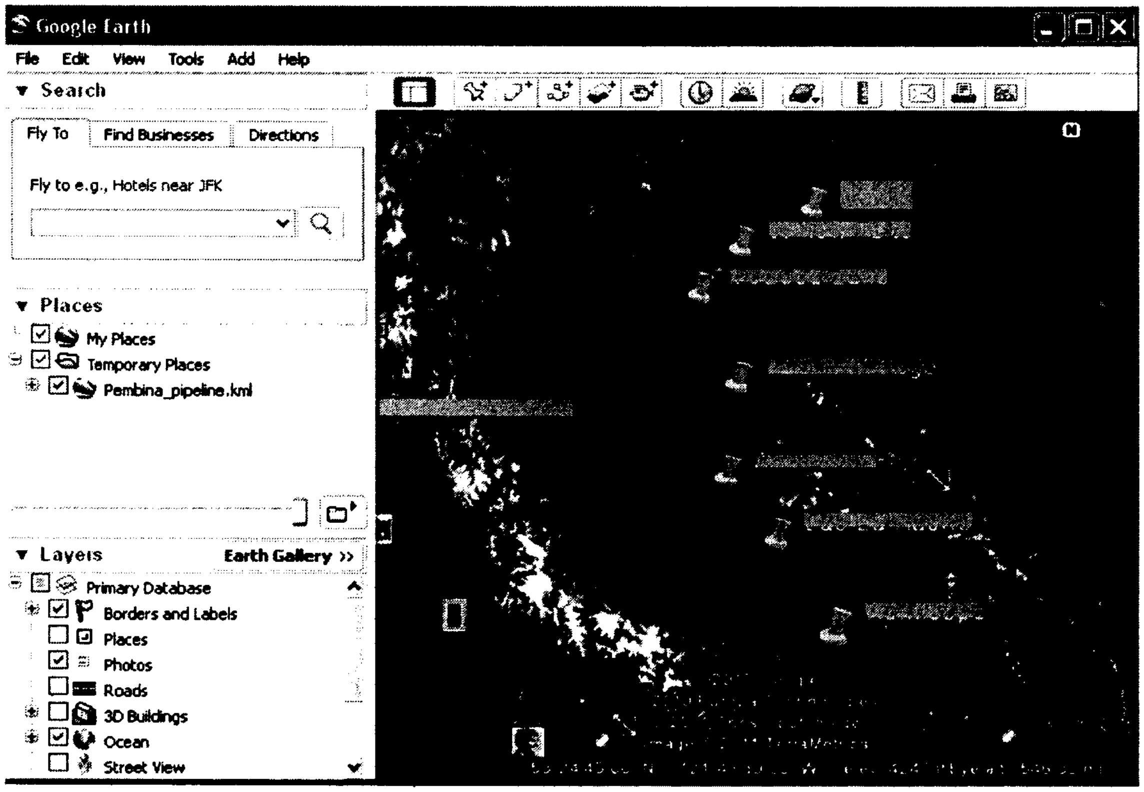


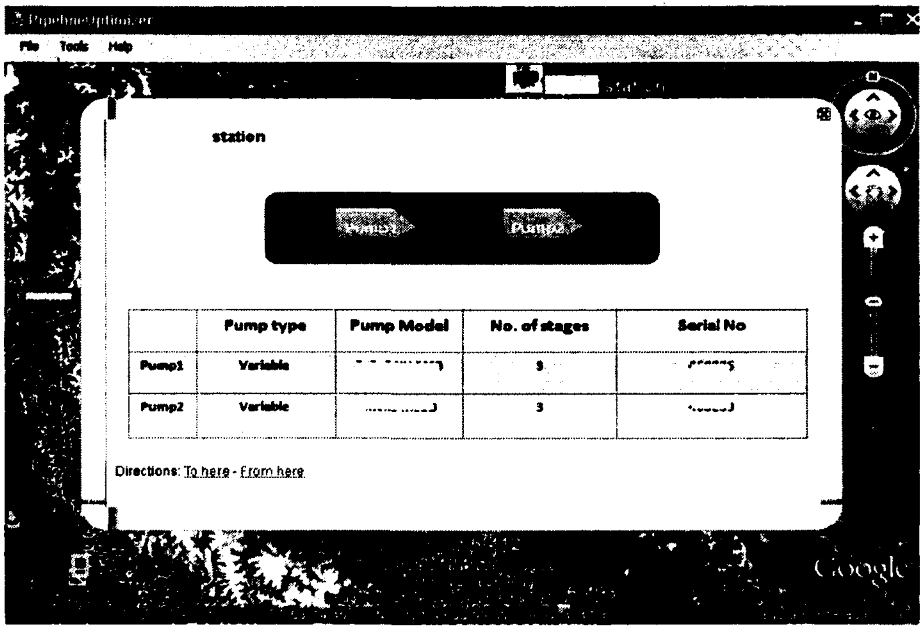
Figure 20: Snapshot of Google Earth application

Google Earth employs a specific textual data file format, called KML (Keyhole Markup Language) to represent different schematics and icons on its graphical interface. KML is an XML notation for representing geographic annotation and visualization within internet-based two-dimensional maps and three-dimensional Earth browsers [148]. A small portion of a sample KML file is shown in Figure 21.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>
<Placemark>
        <name>Pembina Pipeline</name>
        <Style>
                <LineStyle>
                        <color>7f00ff00</color>
                        <width>10</width>
                </LineStyle>
        </Style>
        <LineString>
<extrude>1</extrude>
<tessellate>1</tessellate>
<coordinates>
-120.658,56.155,409.9
-120.648,56.145,473.35
-120.648,56.125,443.17
-121.65,55.708,757.73
-121.66,55.687,625.75
-122.203,55.645,641.7
-122.95,55.125,737.92
-122.97,55.125,709.26
```

Figure 21: Sample KML file showing header information followed by "placemark" tag and sample coordinates used to demonstrate different stations on GoogleEarth

The information related to the geographical locations of each pipeline element, i.e. pump station, pump segments and control values are saved in these files and each time the Google Earth starts up, the information automatically loads on its graphical interface and the user can browse the pipeline. This gives the user a real flavour of where the pipeline is located on the map and the user also can view the internal information related to each pump station, such as the number of pumps and their types, as shown in Figure 22.



station

	Pump type	Pump Model	No. of stages	Serial No
Pump1	Variable	[illegible]	3	[illegible]
Pump2	Variable	[illegible]	3	[illegible]

Directions: To here - From here

Figure 22: S3 station internal information

5.3.4 *Text files*

The optimization solver produces large amount of information after solving the optimization problem. Only a small portion of this information is required to be extracted for further calculations or the creation of charts. The Lindo environment provides the user with the possibility to redirect their data of interest (among optimal values) into text files; therefore text files are where we store our target optimal values. After each optimization run, the content of the text files are replaced with the new optimal values.

In order to be consistent with our optimization data, the historical data taken from our industrial partner is saved in text files. It is worth mentioning that we received huge amount of historical data saved in Excel files from Pembina, where we were required to identify and extract our data of interest. In some cases further data manipulations, such as unit conversion or parameter calculations were also required to convert their data into a proper usable format for our application. This historical data was gathered using Pembina supervisory control and data acquisition (SCADA) system. SCADA systems are used to control and monitor industrial, infrastructure, or facility-based processes.

5.3.5 *XML File*

XML files are typically used to transport and store data. Here in this problem, XML files are used to store the specifications related to the pipeline system, which mainly includes the name and number of pump stations, the number of pump units in each pump station, the path of the Lindo formulation file for the pipeline operation optimization and the path for the KML file containing the geographical profiles of the pipeline.

In order to load a new pipeline and get the system parameters related to each pipeline renewed, the pipeline specification which is stored in a certain XML file is required to be loaded to the system. Thus in order to make the load scenario possible, the user of the system has to first generate the corresponding XML file. A sample XML file used in loading a new pipeline in the system is shown in Figure 23.

5.3.6 *MS Excel*

Ms Excel is used in order to create the charts in this application. The optimization data as well as data taken from SCADA system which are all saved in text files are required to be visualized for comparison and decision making purposes. This is done using the charting feature of Ms Excel because of the high flexibility, support and ease of use that can be achieved by using Excel charts. The idea of using Excel charts becomes stronger when one knows that the chart page in Excel sheets can be simply exported to image files. These image files also can be easily shown in application user interface.

```
<?XML version="1.0" encoding="utf-8"?>
<pipeline pump_stations="5">
        <pump_station id="S1"
                pump_no="2"
                pump_type="variable"
                a="-0.0124"
                 b="0.4903"
                 c="806.5363"
                 const="0.8713"
                 nominal_speed="2900">
         </pump_station>
         <pump_station id="S2"
                 pump_no="2"
                 pump_type="variable"
                 a="-0.0022"
                 b="0.4345"
                 c="926.9063"
                 const="5.1234"
                 nominal_speed="5400">
        </pump_station>
        <pump_station id="S3"
                pump_no="2"
                pump_type="fixed">
        </pump_station>
        <pump_station id="S4"
                pump_no="3"
                pump_type="fixed">
        </pump_station>
        <kml path="\\Pembin_Pipeline.kml"
                path_type="relational">
        </kml>
        <optimization formulation_path="c:\\pembina3.lg4"
                path_type="absolute">
        </optimization>
</pipeline>
```

Figure 23: Sample XML file used for loading a new pipeline

We used Microsoft Component Object Model (COM) technology to interact with Ms Excel from .NET framework. COM is a technology offered for Microsoft Windows-family of operating systems that enables software components to communicate with each other [149]. COM is integrated in several applications such as Microsoft Office Family of products. The .NET Framework provides interoperability with COM, which enables

COM-based applications to use .NET components and .NET applications to use COM components. Employing this technology, we communicated with Ms Excel from our application in .NET framework, to create a chart object using the target optimal and SCADA values imported that chart object to a bitmap image file and then showed the image of the chart in our application.

5.3.7 ***Optimization formulation file***

As explained in Chapter 4, the pipeline operational characteristics and constraints are formulated in a script where the solver can finds the mathematical formulation of all the required elements of the optimization problem, such as decision variables and objective function and find the optimal values considering the configured constraints and parameters. The optimization formulation scrip used in this project is shown partially in Figure 24.

```
! S1 station with two identical pumps H_S11, H_S12 ;
@FOR(Station(s) | s #EQ# 1:
@BIN(B_S11(t)) ;
@BIN(B_S12(t)) ;
@BIN(Te_S1(t)) ;
H_S1_Suc(t) = 1238000/ro/g ;
H_S1_Disch(t) = H_S1_Suc(t) + H_S11(t) + H_S12(t) ;
P_S11(t) = P_Q_S11 * (Q_T(t) - Q_S11(t)) + P_H_S11 * H_S11(t) +
P_Icpt_S11 * B_S11(t) ;
P_S12(t) = P_Q_S12 * (Q_T(t) - Q_S12(t)) + P_H_S12 * H_S12(t) +
P_Icpt_S12 * B_S12(t) ;
H_S11(t) < B_S11(t) * H_Max ;
H_S11(t) > B_S11(t) * H_min ;
H_S12(t) < B_S12(t) * H_Max ;
H_S12(t) > B_S12(t) * H_min ;
Q_S11(t) > (1-B_S11(t)) * Q_min ;
Q_S11(t) < (1-B_S11(t)) * Q_Max ;
Q_S12(t) > (1-B_S12(t)) * Q_min ;
Q_S12(t) < (1-B_S12(t)) * Q_Max ;
C_S11(t) = P_L_S1(t) * Rate_L_S1(t) + P_H_S1(t) * Rate_H_S1(t) ;
P_L_S1(t) + P_H_S1(t) = P_S11(t) + P_S12(t) ;
P_L_S1(t) < P_Thresh_S1 * Te_S1(t) ;
P_H_S1(t) > P_Thresh_S1 * (1-Te_S1(t)) ;
P_H_S1(t) <= 100000 * (1-Te_S1(t)) ;
```

Figure 24: Sample part of the optimization formulation file [139]

5.4 Use-Case Diagram

In previous section, we presented all the actors and systems interacting with our system and described their key roles in the application. In order to represent the system requirements, the main functionality of the system, system actors and their relations are sketched in the use-case diagram. The use case diagram of the system is shown in Figure 25.

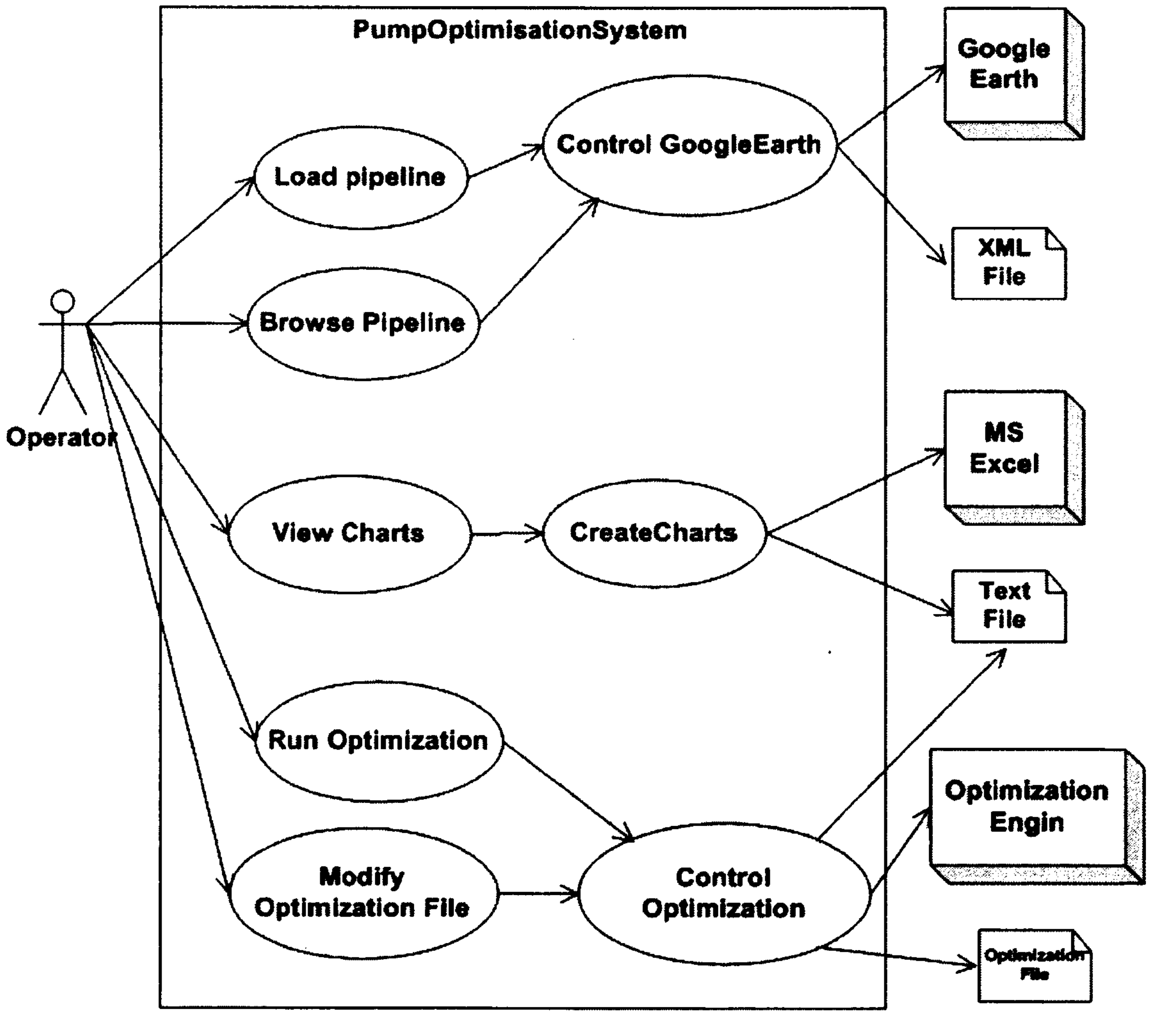


Figure 25: System use-case diagram

5.4.1 *Use case Specifications*

In this section we discuss three of the important use-case specifications of the system.

5.4.1.1 Load pipeline

Brief description

Operator selects and opens a new XML file which contains the specifications for the new pipeline and the system extracts the required information from XML file in order to renew internal parameters related to pipelines and also the path for KML file which includes the geographical profile of the new pipeline.

Basic Flow of events

1- The operator clicks on the "Load new pipeline" submenu from "File" menu.

2- The system displays the "open dialogue box" for browsing and selecting the XML file related to new pipeline.

3- The operator selects the specific XML file and click on the "open" button.

4- The system reads the required information from the XML file and renews the system internal parameters and loads the pipeline schematic on Google Earth interface accordingly.

Alternative flows

- Invalid XML file

If in step 3, the user selects a wrong file, either not having a standard XML format or not having all the required information for the system to reload the pipeline related parameters or not having a valid path to read a KML file, the use case ends with a failure condition and the system shows the proper error message.

- Invalid KML file

If the Google earth does not find the KML file having a standard format, it is not possible for Google Earth to load the pipeline on its graphical interface. The Interface will remain blank and the system shows a proper error message.

Preconditions

- The user must be in the application main form (all the modal windows must be close.).
- Google earth is installed on the computer running the application.

Post conditions

- Successful Completion

The user can start browsing the new pipeline.

- Failure Condition

The Google Earth will remain blank. The user can either select and load a new pipeline or choose to close the Google Earth application and use other features of the system.

Extension points

In step 4, while loading the KML file on Google Earth, the "manage Google Earth" use-case will be referenced and used in order to load a new instance of Google Earth. This is required to clear up any previously loaded pipeline on the Google Earth graphical interface.

5.4.1.2 View charts

Brief description

Operator chooses to view the optimization charts.

Basic Flow of events

1- Operator chooses the "view chart" submenu from "Tool" menu, 2- Operator chooses the station name in the "chart" modal window,

3- Operator also chooses the chart type (either operation cost or pumps speed),

4- The system reads the related SCADA and optimization data from corresponding files,

5- The system passes the data to Ms Excel,

6- The system asks Ms Excel to generate the Excel charts,

7- MS Excel converts the chart to a bitmap image and returns the image path to the system,

8- The system shows the resulting chart image.

Alternative flows

1-Ms Excel is busy with another application

Ms excel does not respond, therefore the system can not pass the target SCADA and optimal values to Excel and the charts can not generated. A proper error message is shown by the system.

Preconditions

- Ms Excel is installed on the computer running the system.
- Files containing SCADA and optimal values should exist in the target directory.

Post conditions

Successful Completion

The user can start viewing different chart types sketched with the SCADA and optimal values taken from different stations.

Failure Condition

The user can view the last successfully built charts (if any).

Extension points

"Create chart" use-case will be referenced and used in order to handle the interactions between the system and Ms Excel.

5.4.1.3 Run optimization

Brief description

The operator calls Lindo to solve the pipeline formulation file.

Basic Flow of events

1-The operator clicks on the "Run optimization" submenu from "tool" menu.

2- The system displays the modal window for selecting the target optimization file and running the solver.

3- The operator selects the target optimization file.

4- The system loads the file.

5- The user can modify the optimization file and then click the save button.

6- The system saves the optimization file.

7- The user click on "Run optimization" button.

8- The system calls the solver and gives the target optimization file to the solver to be optimized. The optimal values (if feasible) are found and saved in text files.

Alternative flows

- Optimization solver is busy with other applications

The system creates a time out error message.

- The formulation file is invalid

A proper error message is generated and shown to the user.

- The optimization problem is not feasible with current variable setting

The system waits for a certain amount of time (currently set to 10 seconds based on the experience) and then checks the output status file and if finds the infeasible status on that, creates and shows a proper message.

Preconditions

Lindo is installed and registered on the computer running the system.

Post conditions

Successful Completion

The files containing optimal values for each station are updated with the new optimal values.

Failure Condition

The files containing optimal values for each station are not updated. The user can change the optimization file variable setting and try to run the optimization solver on the new file again.

Extension points

The "Manage optimization" use-case will be referenced and used to manage the interactions between the system and the optimization solver.

5.5 **Activity Diagrams**

In order to represent the stepwise actions and workflow of the components of the system, we have used activity diagrams. In this section we present three of the main activity diagrams of the system use-cases.

5.5.1 ***Run Optimization***

Based on the information presented in the use-case specification of "calling the optimization solver", the activity diagram for this use case is shown in Figure 26.

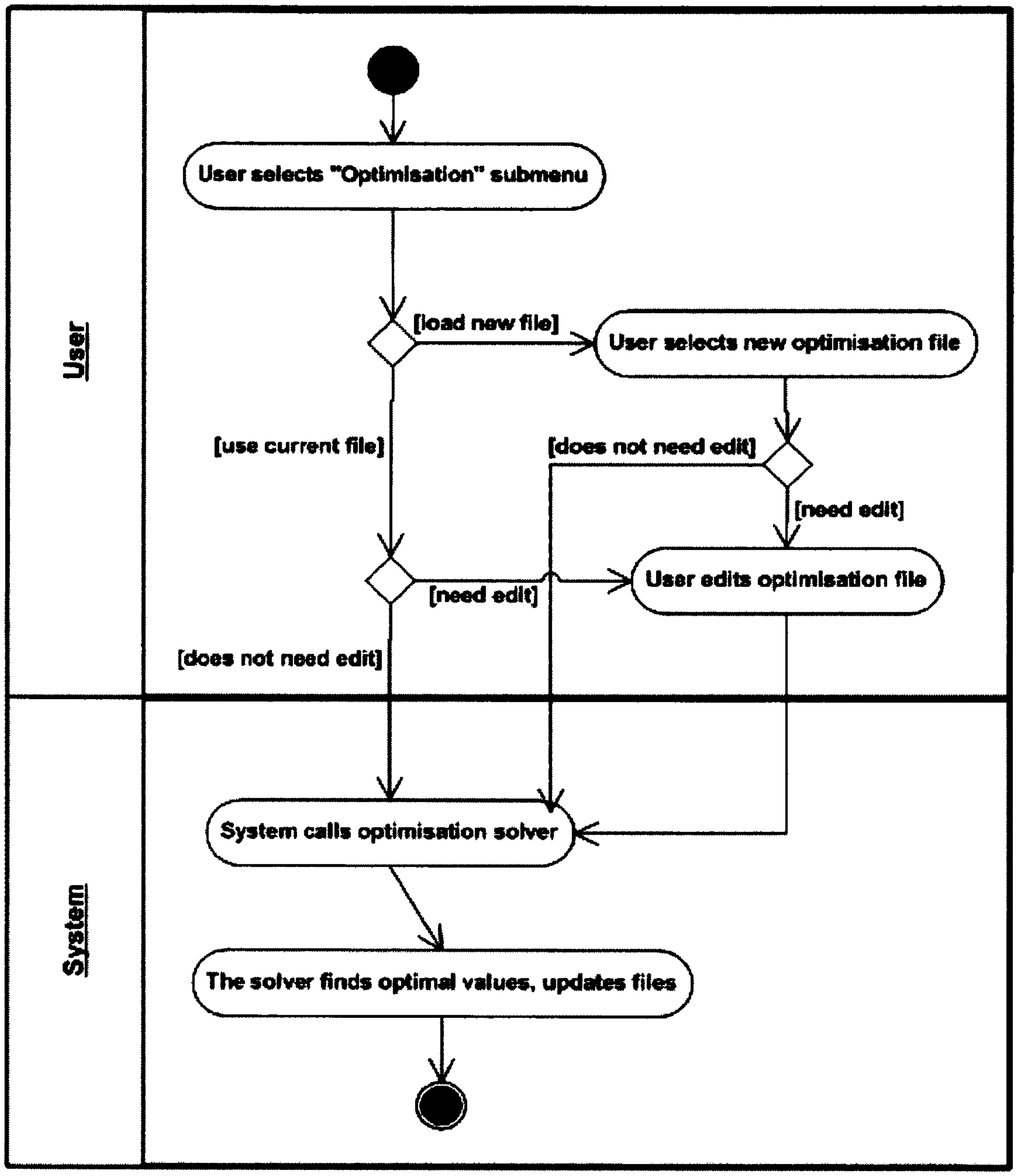

User
System
User selects "Optimisation" submenu
[load new file]
User selects new optimisation file
[use current file]
[does not need edit]
[need edit]
[need edit]
User edits optimisation file
[does not need edit]
System calls optimisation solver
The solver finds optimal values, updates files

Figure 26: Activity diagram for "Run optimization" use-case

As seen in the figure, different steps and conditions of the use case are graphically represented.

5.5.2 *View Charts*

The activity diagram for viewing the charts is shown in Figure 27.

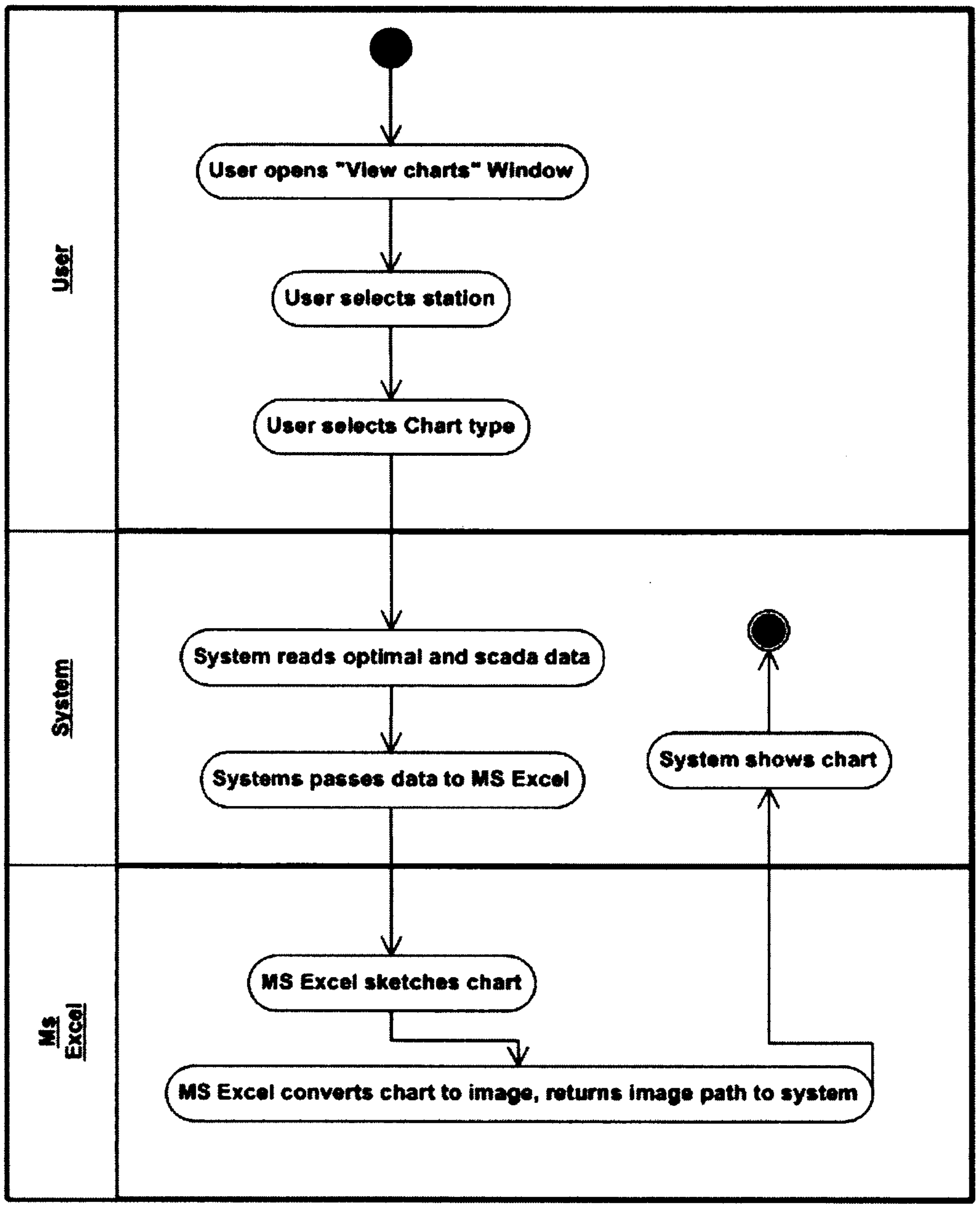

User
System
Ms Excel
User opens "View charts" Window
User selects station
User selects Chart type
System reads optimal and scada data
Systems passes data to MS Excel
System shows chart
MS Excel sketches chart
MS Excel converts chart to image, returns image path to system

Figure 27: Activity diagram for "View chart" use-case

5.5.3 *Load Pipeline*

The activity diagram for loading a new pipeline is shown in Figure 28.

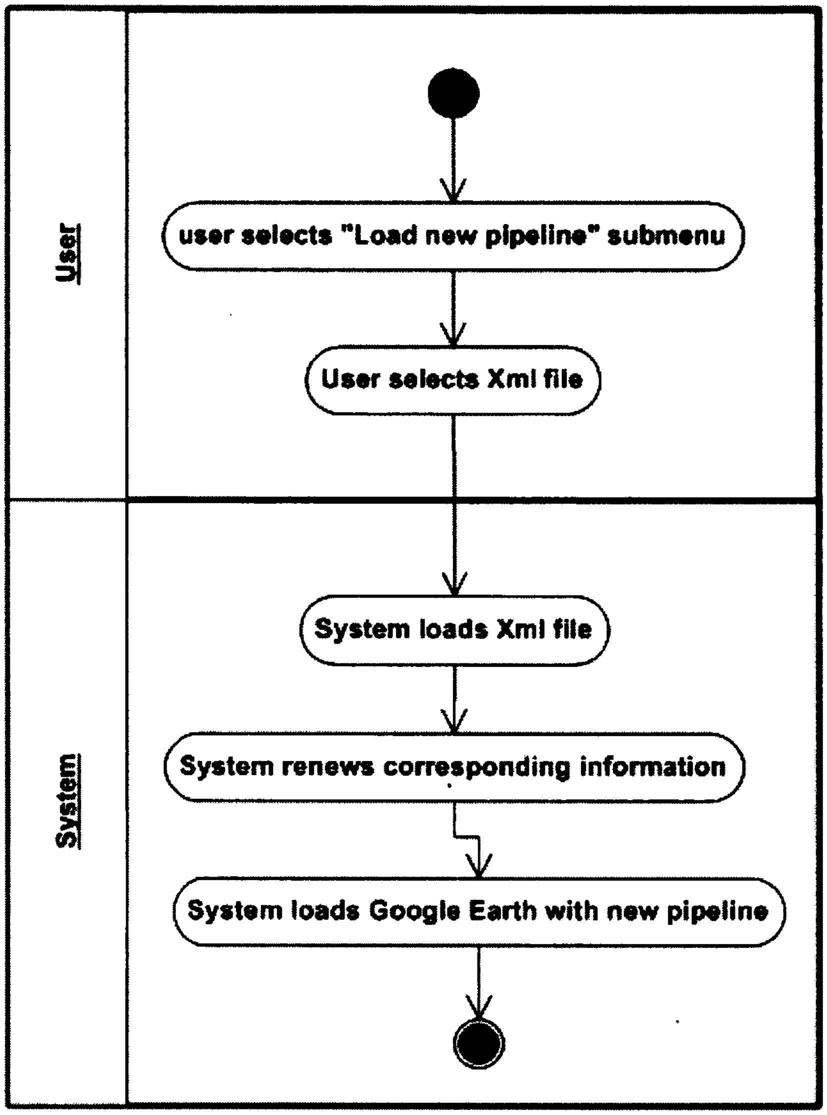

User
System
user selects "Load new pipeline" submenu
User selects Xml file
System loads Xml file
System renews corresponding information
System loads Google Earth with new pipeline

Figure 28: Activity diagram for "Load pipeline" use-case

5.6 Architecture: MVC

In order for supporting testability, maintainability and separation of logics and concerns, the MVC pattern was introduced back in 1979 and first implemented in Smalltalk-80 [150].

Design for testability has not gained any attention in scientific software development according to the primary studies we had in our systematic review. We could not find any publication in which the testability had been a concern of the SES designers. Therefore to bring testability into our attention, while we were considering the issues of design, we have adopted Model-View-Controller (MVC) architectural pattern in our system in order to better manage different levels of the system.

Class type	Class name	Description
Controller	mainController	The controller which manages the main functionality of the system, such as creating chart, loading a new pipeline and populating the corresponding network with the data taken from optimization and SCADA text files
	GEController	The controller which manages the system's interactions with Google Earth, such as moving in different directions and zooming in and out
	optimizationController	The controller which manages the system's interactions with optimization engine, such as running the optimization solver and opening the optimization editor
View	mainForm	The main window of the application which contains GoogleEarth and menus
	chartForm	The window for creating and reviewing different charts
	optimizationForm	The window for editing the optimization formulation file parameters
Model (Entity)	Network	Class representing a pipeline entity
	pumpStation	Class representing oil pump station entity
	Pump	Class representing pump entity
	GE	Class used to represent a Google Earth instance
	Lindo	Class used to represent a Lindo (optimization solver) instance
	ExcelApp	Class used to represent an "Excel application" object used to create a chart

Table 20: System classes categorization based on MVC architecture

In MVC architecture the system is broken down into three components: model, view and controller. The model component is the application's business layer and usually includes the objects that represent the business entities which make up the application such as pump units and power stations.

The view layer is the application's user interface and consists of different modal and modeless windows and other standard GUI elements such as menu, buttons, images, radio buttons and panels. The choice of having an isolated layer for all the interactions with the user will later help us in the testing graphical user interface of the system.

The controller layer is where all the events, generated by user-interface actions, such as when the user clicking a button or selecting an item from a drop-down list are processed by the application. Different elements of the system are shown in Table 20.

5.7 **Class Diagram**

Class diagram is one of the static structure diagrams that demonstrate the structure of the system using the classes of the system, their attributes, their methods and the relationships between the classes [151]. The simplified class diagram of the system is shown in Figure 29. In this figure classes are color coded for better separation of different layers of MVC. Forms are coloured in yellow, controllers in grey and entities are represented in green. Setters and getters (accessor of network, pump and pumpStation are not shown for the sake of simplicity.

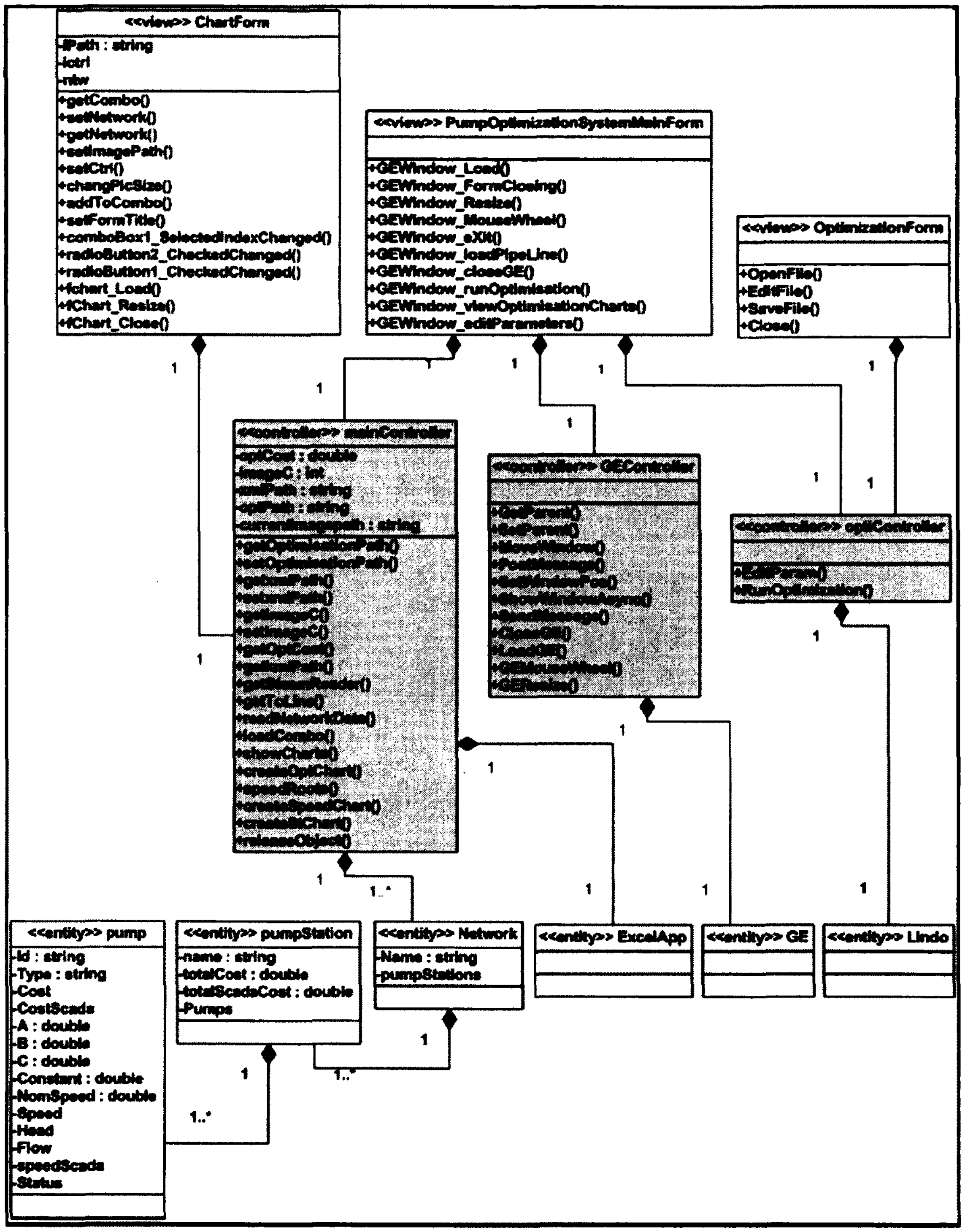


Figure 29: Application class diagram

5.8 Discussion

In this Chapter, the requirements of the system besides its analysis and design were described. In response to H1.1 (first hypothesis of the case study first research question: Gaining domain expertise is time-consuming and difficult for software engineer, compared to learning the basics of typical non-scientific/non-engineering domains), besides the findings of the SLR presented in Chapter 3, we also observed that understanding the problem domain and application requirements is one of the main challenges in SES development [33]. Principles and practices entangled with scientific and engineering domains are usually complex and hard to understand for software engineers, not having any related experience and background. Our experience in the requirement elicitation of the application confirms that gaining the domain expertise is tedious and time consuming for us as software engineers, because of the certain complexities of the pipeline operation.

Regarding H1.2 (second hypothesis of the case study first research question: The requirements cannot be decided in early stages of the development as they evolve throughout the process), our experience matches with that of the literature; we observed that the requirement specifications can not be finalized in early stages of the development, and this is an ongoing process in the real world as software engineers and domain experts need to learn how to communicate. Another observation we had was that most of the time domain experts did not have a clear understanding of how the software could be integrated with their every day routines and they did not know how it could be utilized in an efficient way beside their own system.

To tackle this issue and in response to H2.1 (first hypothesis of the case study second research question: Regular meetings with domain experts is a very beneficial practice for validation of the requirements), as suggested in the literature [63], in our case study, meetings were organized with the industrial correspondent to frequently verify our understanding of the pipeline operation principles, documenting the domain concepts and basics as required and verifying the system requirements.

Object-oriented methodology is not very common in SES community, as the scientists and engineers do not have the required background to benefit from this methodology [34]. In our case study in response to H2.3 (third hypothesis of the case study second research question: Adopting OO methodology, can pave the way for using design and architectural patterns besides giving a better management over data and functions), object-oriented technology was adopted in designing the system to build a robust framework, as suggested in [127], and to better manage and modify different modules of the application.

Employing design and architectural patterns have shown to provide remarkable advantages for reusability and maintainability of SES [116]. We used MVC architectural pattern to bring testability and maintainability to our system, in order to address these non-functional requirements of the target software. According to the results of SLR, testability in particular is a factor which is often ignored in SES design. As testing SES is considered complex and tedious according to literature, we tried to incorporating testability in system design.

5.9 Chapter Summary

This chapter summarises our oil pipeline operation application requirements as well as providing analysis and design documents used in designing the system. We adopted object-oriented methodology in this study. System actors, use-case diagram and some important activity diagrams such as “calling the optimization solver”, “viewing charts” and “loading a new pipeline” are presented.

We followed MVC architectural pattern in order to support testability besides better managing levels of our application. Different elements of the system based on MVC architecture, which include the system entities such as pump stations, controllers such as optimization controller and the Google Earth controller and application views, which are different windows of the system are tabulated and described.

The process of application development will be discussed in next Chapter, and the dependency analysis of the system artefacts will be discussed.

Chapter Six: **Development**

In this chapter, we present the details on the implementation and development phase of the pipeline operation application. For our development platform, we used Visual studio 2008. C# programming language was chosen for implementation, mostly because we were proficient in it and our industrial partner preferred it to avoid further inter-operability issues with their other applications.

We start by elaborating our development process in Section 6.1, which was the iterative development approach, followed by dependency analysis in Section 6.2, to gain better understanding about the characteristics and dependencies of the code. This type of analysis and visualization brings the developer a better understanding of the code and makes the code maintenance easier and more cost-effective. Section 6.3 provides the lessons learned and concludes this Chapter.

6.1 **Development Process: Iterative Approach**

According to the certain characteristics of SES, as mentioned in Chapter 3, we decided to take an iterative development approach in our oil pipeline application development. This model is depicted in Figure 30. The process starts with initial planning for the development, followed by preliminary understanding of the system's requirements, analysis and design, feature implementation, deployment, testing and primary evaluation. Next iterations complement the activities of previous iterations by adding new features and functionalities, may raise the need for redesign and refactoring. This gives the developer the opportunity to benefit from what they learned in previous iterations for improving the development quality.

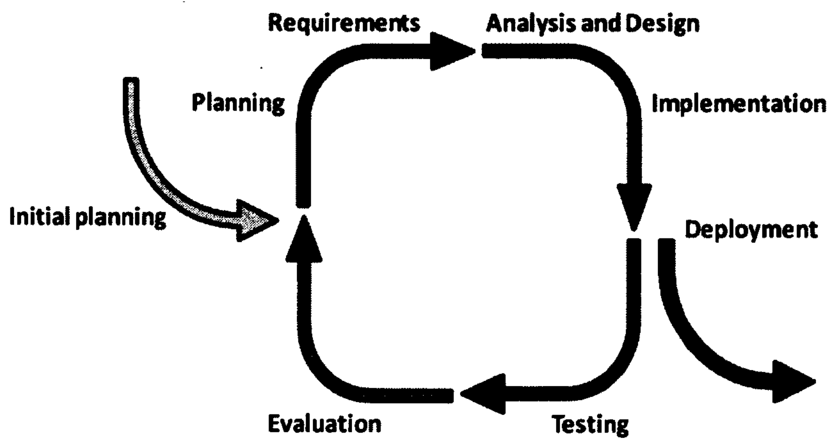


Figure 30: A model of Iterative development approach [152]

According to the results of our SLR, this approach works well with the nature of SES, as it gives flexibility to the requirement elicitation and provides the opportunity of getting iterative feedback from our industrial partner to evaluate the understanding of the problem, on a regular basis. This evaluation is considered very critical to the development to avoid misunderstanding of the concepts, as they are not primarily expert in the pure science they develop the software for.

In this study, as mentioned in Chapter 3, we started by discussing with our industrial partner to identify the requirements of the system they needed. We faced challenges in understanding their language, which were settled after discussing the details over several meetings and learning their specific domain terminologies.

The analysis and design then took place with the appropriate choice of object-oriented methodology, followed by partial feature implementation and testing of the system. The system then was evaluated based on our industrial partner's expectations of necessary functionalities as stated in the requirements section, besides our own understanding of what might be helpful for them after having several meeting with the domain experts and investigating similar pipeline operation software. Features were prioritized roughly based on their importance with principle investigator (advisor of this thesis) in the beginning of the project and new features were gradually added in the following iterations. Sometimes it was required to break a complicated feature into several smaller tasks and then develop each of those smaller tasks during one iteration. This way the development of such a feature took several iterations before it can be fully integrated into the system. Manual evaluation and automated testing based on the proper functionality of the system were held regularly between the author and the principle investigator in their weekly meetings.

6.2 **Dependency Analysis**

In this section we briefly present some ideas which help reuse and maintenance of the application. According to the results of our SLR, the maintenance stage of SES development has not yet gained enough attention and reusability of the software is often ignored. The ideas discussed in this section suggest improving the maintenance quality.

In order to understand a piece of code and judge about its quality for re-use purposes, it is required to study what it depends on and also what depends on it [153]. If the component under study is found to have a large group of dependencies on other systems or components, then it will potentially change whenever one of those other systems or components change. Dependency analysis is performed to identify and understand the existing dependencies between code in order for managing the complexities that may arise as the result of changes and updates on the code.

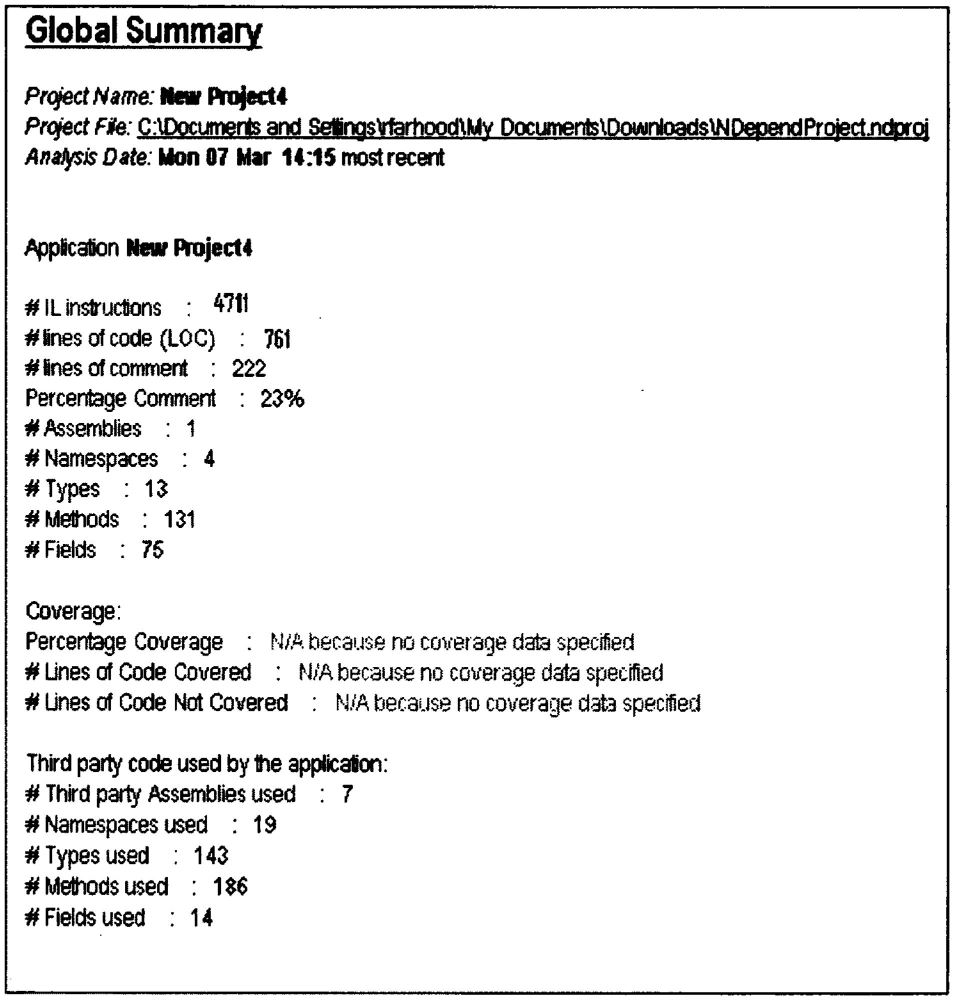
To analyse the code dependencies within a system, all the existing relationships along with the source and the target of those relationships should be identified [153]. These relationships are also possible to be dependent on other systems or components, therefore all the indirect dependencies are required to be identified and traced within the system. There is a graph formalism called the dependency graph, which includes all the existing relations among the code. In the following section, we demonstrate the dependency graph for our application and briefly introduce the tool used to perform the dependency analysis in our study.

6.2.1 *Applications Metrics*

We used Ndepend [154] to calculate some of the application metrics, as discussed in this section. Ndepend is a popular tool that can be integrated with Visual Studio and provide dependency analysis utility as well as metrics calculation.

After creating a project in Ndepend by selecting the target assembly to be analyzed, the user can run the analysis and then study the reported results. Ndepend investigates the code and create several reports on the application statistics and metrics.

One of the application's metrics reported by Ndepend is the number of Intermediate Language (IL) instructions. When the C# code written in .NET Framework is compiled, the compiler generates assemblies which contain byte-code. In the .NET framework, an assembly is a group of types and resources that builds a logical unit of functionality and is usually used for deployment, versioning, and security purposes. Assemblies are stored as .exe or .dll files [155]. These assemblies can then be executed by Common Language Runtime (CLR) which is the engine for code execution in .NET Framework. The byte code is called IL. The number of IL instructions in a system is considered a size measure, which can be determined just after the source code is compiled.



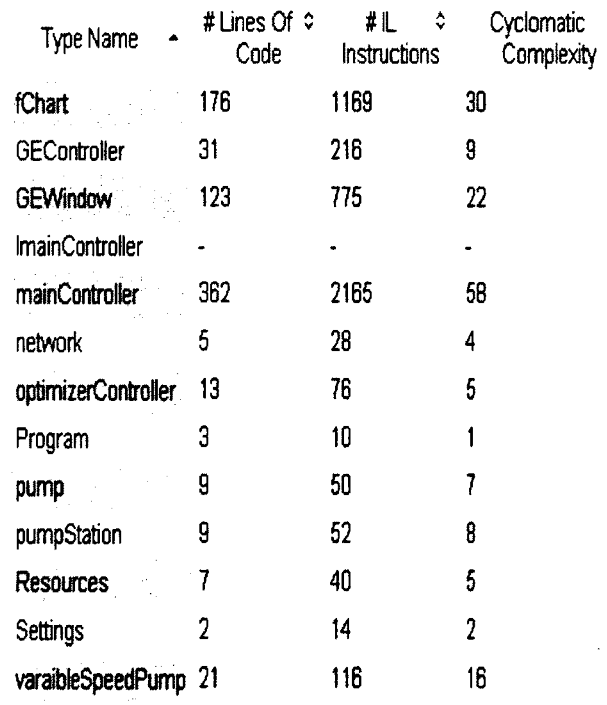
```
Global Summary

Project Name: New Project4
Project File: C:\Documents and Settings\rfarhood\My Documents\Downloads\NDependProject.ndproj
Analysis Date: Mon 07 Mar 14:15 most recent


Application New Project4

# IL instructions  :  4711
# lines of code (LOC)  :  761
# lines of comment  :  222
Percentage Comment  :  23%
# Assemblies  :  1
# Namespaces  :  4
# Types  :  13
# Methods  :  131
# Fields  :  75

Coverage:
Percentage Coverage  :  N/A because no coverage data specified
# Lines of Code Covered  :  N/A because no coverage data specified
# Lines of Code Not Covered  :  N/A because no coverage data specified

Third party code used by the application:
# Third party Assemblies used  :  7
# Namespaces used  :  19
# Types used  :  143
# Methods used  :  186
# Fields used  :  14
```

Figure 31: Snapshot taken from NDepened analysis report

Number of lines of code, as reported by Ndepend is different from physical LOC (which is calculated by counting application's lines of source code). This metric which is referred to as Logical LOC, is calculated in Ndepend by the information taken from PDB files. A PDB or Program DataBase file contains information related to the debugging of the application and the project state [154]. The logical LOC for a method is then calculated by counting the number of sequence points for that method in the PDB file.

Sequence points are used to highlight a spot in the IL code that corresponds to a certain location (usually a part of a statement) in the original code.

As it is demonstrated in Figure 31, IL instructions, logical LOC, lines of comment, number of used assemblies, namespaces, types, methods and fields are calculated and shown. Also there is some information on third party code used by the application, as shown in the snapshot, which refers to the code referenced by our application's assembly and source code. Brief description about this third party code can be found in Table 22.



Type Name ▲	# Lines Of Code ⇕	# IL Instructions ⇕	Cyclomatic Complexity
fChart	176	1169	30
GEController	31	216	9
GEWindow	123	775	22
ImainController	-	-	-
mainController	362	2165	58
network	5	28	4
optimizerController	13	76	5
Program	3	10	1
pump	9	50	7
pumpStation	9	52	8
Resources	7	40	5
Settings	2	14	2
varaibleSpeedPump	21	116	16

Figure 32: Application classes' main metrics breakdown

In Figure 32, the breakdown of the main metrics of our application is tabulated. Cyclomatic complexity metric values are also calculated and shown for the application classes. This complexity metric shows the number of linearly independent paths through the program source code [156], which can be calculated using the control flow graphs. MainController class, fChart (which is the class for the view chart window) and GEWindow (which is the class for the main window of the system) have higher cyclomatic complexity values compared to other classes of the system, as the main functionality of the application are embedded in them. This distribution of complexity is enforced to different classes of the system as the result of using the MVC architecture.

6.2.2 ***Dependency Graphs***

We used Ndepend [154] to perform the dependency analysis. As explained earlier in this chapter, dependency graph demonstrates all the existing dependencies among system's elements. The dependency can be identified in different levels, such as namespace level, class level and method level.

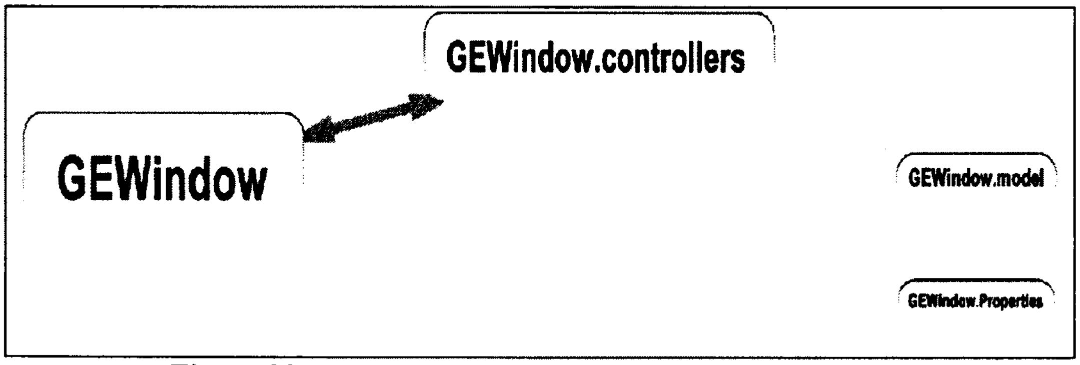


Figure 33: Dependency graph, system namespace level

For example Figure 33, demonstrates the dependencies between the application's namespaces. In .NET framework, namespaces are used to group the type names in order to reduce the chance of name collisions [155]. The thickness of the edges connecting different boxes in the dependency graph is proportional to the degree of coupling between those entities. Here this means that the edge thickness connecting the two namespaces is proportional to the number of classes of the source namespace which are using the classes of the target namespace added to the number of classes of the target namespace used by the source namespace. As seen in the figure, there exists a noticeable strong dependency between the application and the controllers compared to the dependency between controllers and model or the dependency between application and model, as the controllers are in charge of controlling the interactions between models and views as well as handling and implementing the business logic in the system.

Looking from architectural level, this is the implication of using MVC architectural pattern, as according to the MVC architecture, the main functionality (business logic) of the system is integrated in the controllers and the user achieves this functionality by interacting with the system through the provided views (forms). Figure 34, presents the dependencies among the mainController class methods. Brief description of these methods is presented in Table 21. The demonstrated relationships show the methods that are being called from within the other methods. For example showCharts() method, calls creatOptChart(), loadCombo() and createStChart(). This gives the reader a clear understanding of the source code implementation within mainController. The dependency graphs for other controllers of the application are presented in Appendix B.

Method Name	Description
getOptimizationPath	Returns the optimization formulation file path.
setOptimisztionPath	Extracts the optimization formulation file path from the corresponding XML file whenever a new pipeline is loaded to the system
getXMLPath	The getter method for the XML path member variable
setXMLPath	The setter method for the XML path member variable
getImageC	The getter method for the image index created by the MS Excel charting utility
setImageC	The setter method for the image index created by the MS Excel charting utility
getOptCost	Returns the total cost of the optimized operation
getKMLPath	Extracts the KML file path from the corresponding XML file whenever a new pipeline is loaded to the system
getSteamReader	Returns a stream reader by which the corresponding text file can be accessed
getToLine	Reads the specified text file lines up to the specified line
readNetworkData	Populates the network object with the corresponding data taken from the text files whenever a new pipeline is loaded to the system
loadCombo	Dynamically re-loads the combo box content (on the view chart window) according to the options chosen by the user
showCharts	Shows the charts according to the options set by the user in the view chart window
createOptChart	Generates the MSExcel charts demonstrating the optimal and SCADA pipeline operation cost daily values
speedRoots	Calculates the roots of a quadratic equation designated for calculating the speed of a variable speed pump based on a group of constants corresponding the each pump, the amount of head generated by the pump and the current flow rate passing through the pump
createSpeedChart	Generates the charts demonstrating the hourly speed of each of the variable speed pumps
createStChart	Generates the charts demonstrating the hourly cost required to operate each of the pumps
releaseObject	Releases the objects created while using the charting utility in MS Excel

Table 21: mainController class methods

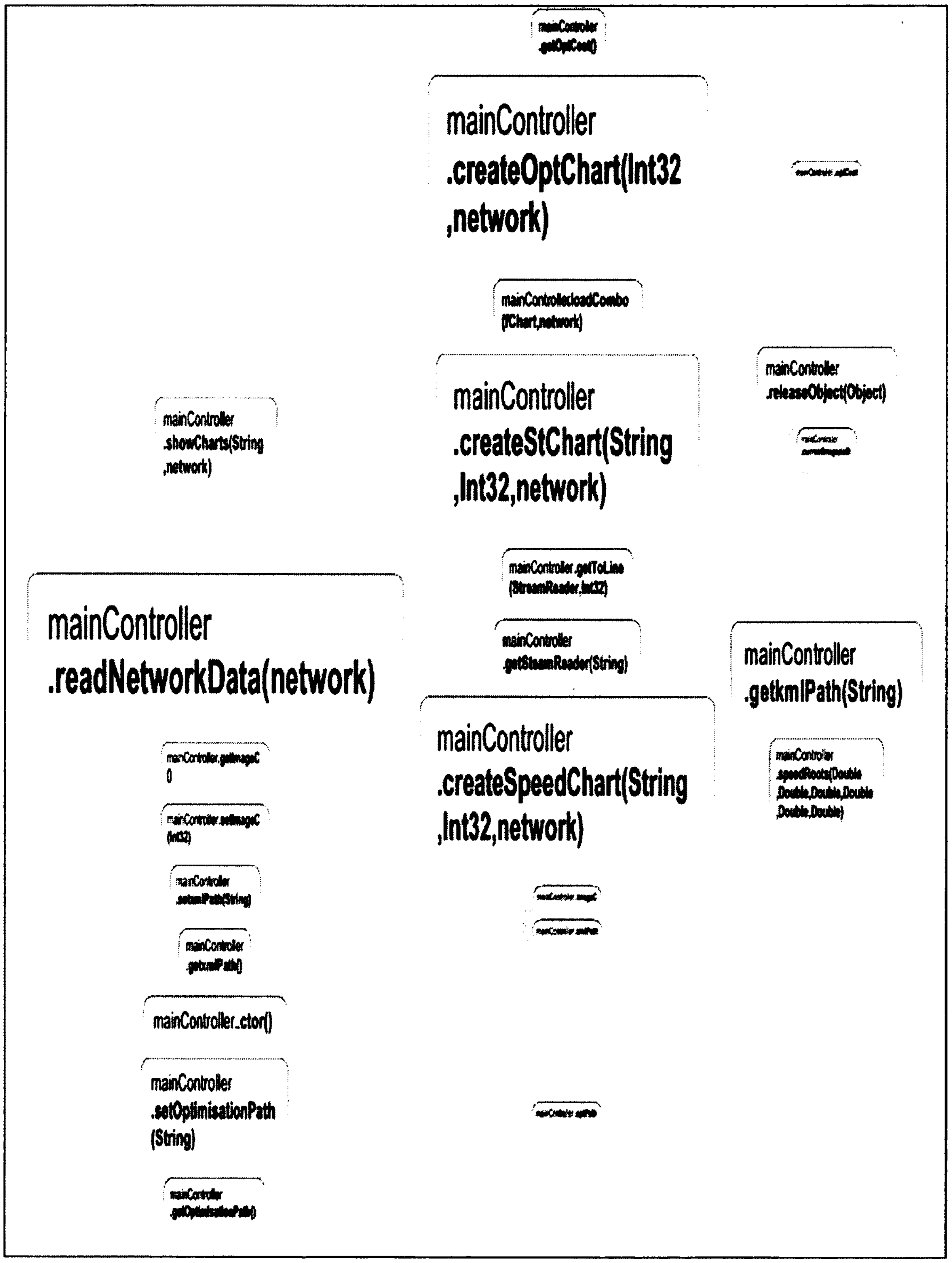

mainController .getOptCost()
mainController .createOptChart(Int32 ,network)
mainControllerloadCombo (fChart,network)
mainController .createStChart(String ,Int32,network)
mainController .releaseObject(Object)
mainController .showCharts(String ,network)
mainController.getToLine (StreamReader,Int32)
mainController .getSteamReader(String)
mainController .readNetworkData(network)
mainController .getkmlPath(String)
mainController .createSpeedChart(String ,Int32,network)
mainController .speedRoots(Double ,Double,Double,Double ,Double,Double)
mainController.getImageC ()
mainController.setImageC (Int32)
mainController .setkmlPath(String)
mainController .getkmlPath()
mainController..ctor()
mainController .setOptimisationPath (String)
mainController .getOptimisationPath()

Figure 34: Dependency graph, within mainController

In the dependency graph shown in Figure 35, the dependencies between our application's assembly (the box labeled GEWindow) and other used assemblies or namespaces are presented. According to the figure our system is highly coupled with System.Winows.Forms namespace. This is mainly because GEWindow is a Windows-based application and System.Winows.Forms namespace contains classes which provide user interface features for Windows-based applications.

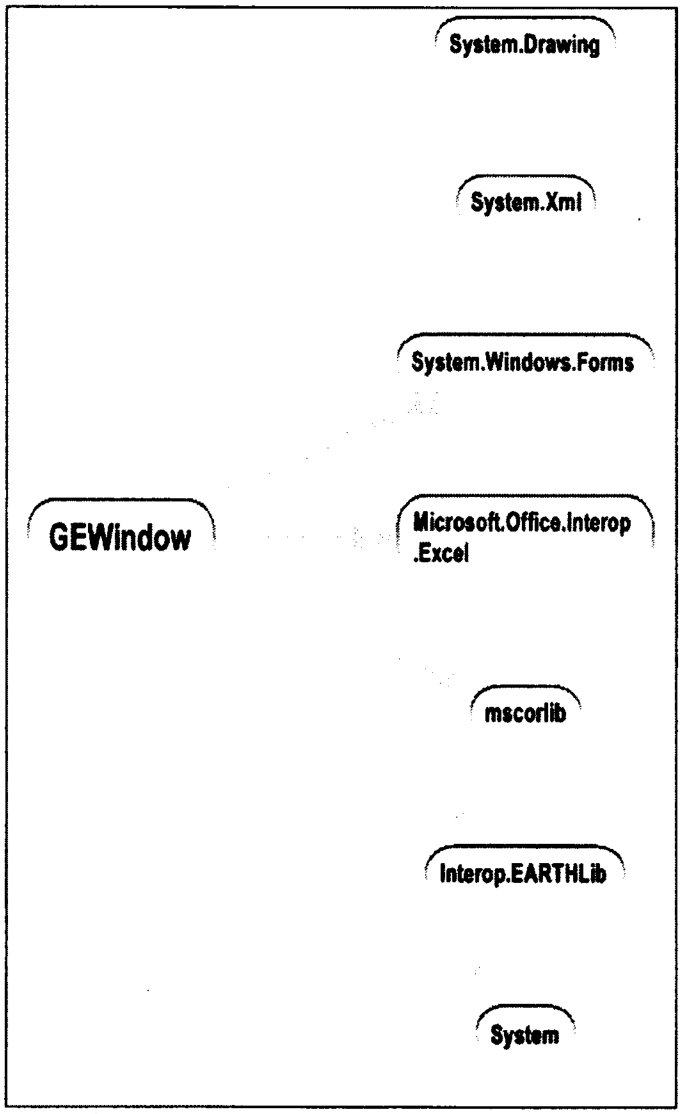


Figure 35: Dependency graph between .Net assemblies

The dependency between GEWindow and mscorlib is also noticeable as this assembly contains base class libraries of .NET framework. A brief description of the assemblies used in our system is presented in Table 22.

More details on the applications statistics and other analysis results performed using Ndepened, are presented in Appendix B.

Assembly name	Description
System.Drawing	This namespace provides access to GDI+ basic graphics functionality [157]. (Windows GDI+ is a class-based API intended to be used by C/C++ programmers which enables applications to use formatted text and graphics on both the video display and the printer [158]).
System.XML	The namespace offers standards-based support for processing XML, such as XML 1.0 and XSD Schemas [157].
System.Winows.Forms	This namespace contains classes for building Windows-based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system [157].
Microsoft.Office.Interop.Excel	This assembly, which belongs to the family of Component Object Model (COM) Interop assemblies, allows unmanaged (COM) code to be called from managed (.NET) code by using the Microsoft .NET Framework and the common language runtime (CLR) [159].
mscorlib	The mscorlib.dll is a shared assembly, which includes the important base class libraries of .Net framework. Applications written for the .NET framework are executed in the software mscorlib.dll to manage the program's runtime requirements [160].
Interop.EARTHLib	Another assembly from COM family to manage the application's interaction with Google Earth.
System	This assembly is a reusable and self-describing building block of common language runtime applications [158].

Table 22: Description of the external assemblies and namespaces used in our application

6.3 Discussion

According to the emerging nature of the requirements in SES [79], they can not be fixed in the beginning of the development. This can be later the source of further challenges in the development [33]. In order to provide evidence in response to our case study first research question, our experience in the implementation and development of the system confirms that requirement have an emerging nature throughout the development; thus, we chose to adopt the iterative development approach, as suggested in [90], also in response to H2.2 (second hypothesis of case study second question: Adopting iterative approach fits the "evolving and emerging requirement" nature of engineering software). Defining iterations for the development introduces the flexibility of bringing in the newly defined requirements at later iterations to the software system. In iterative approach in each iteration, the most important requirements are taken into account and integrated into the system.

Maintenance difficulties and the need for long-term maintenance are mentioned to be challenging in SES development according to the literature. We performed dependency analysis in order to extract the dependencies among different system artefacts. Regarding the case study second research question, this was conducted to help further maintenance of the system artefacts. Having dependencies demonstrated as the result of dependency analysis, the elements which are dependent on the changing elements or the elements which are being referenced by the changing elements can be easily identified for any potential need for update or change.

6.4 Chapter Summary

In this chapter, the iterative development process used to develop the system under study was introduced. Dependency analysis, in order to extract the dependencies among the system artefacts, was performed. This analysis identifies the highly dependent elements, whish later gives the developer a precise idea about the element relationships, and is helpful when any artefact of the application is required to be re-used, upgraded or replaced. Some of the application metrics such as lines of code, IL instructions and cyclomatic complexity was calculated and reported followed by selected dependency graphs.

Next Chapter will discuss the testing approach used to test the functionality of the system as well as testing the GUI.

Chapter Seven: **Testing**

The quality of the software systems need to be systematically checked to assure those systems meet the requirements and specifications. Testing SES in specific, as mentioned in Chapter 3, is a twofold challenge; testing the scientific/engineering solution offered to address a scientific or engineering problem besides testing the software which is designed to utilize that solution in a proper and easy way. To be more specific in the context of software engineering, by testing we mean investigating if a program is behaving as expected [161]. Here, we aim at automated testing compared to manual testing, in which a program or application is written to exercise and verify (assert) Software Under Test (SUT). To be more specific, here we automatically run manually written tests.

In this chapter we discuss the automated testing strategies we undertook, in order to test the software functionalities. To achieve this goal, we planned to perform automated unit testing and automated GUI testing of the system as two of the most common and standard testing techniques used in software projects to detect bugs. As mentioned in Chapter 6, we adopted the iterative development approach in which testing is a part of each iteration. Therefore, in each iteration, the implemented features were tested in addition to testing the previous code to make sure nothing was unintentionally broken by adding the new features.

As mentioned in Chapter 5 on the analysis and the design of the system, we used MVC architectural design pattern in order to classify and separate the concepts in the system in order to make the testing process more straight and manageable. In MVC architecture, the business logic is separated from view and presentation logic, which makes testing of the functionality independent from testing the view layer [158, 162]. As shown in

Table 20, the system consists of 3 controllers, each of which is responsible for a group of similar functionalities provided by the software. For example all the methods and variables required for implementing the interactions and functionalities related to Google Earth in the system and working with that are placed in GEController. This separation and classification of functionalities and concepts, as will be discussed in the next sections, assisted us in figuring out which classes contain the code that is more likely to change or break during the development of the new features and which classes does not require being included in the testing cycles, mostly because they are using adequately tested libraries.

For the verification of the accuracy of the optimization model, sensitivity analysis on some of the important pipeline operation parameters is conducted by another student in our group, who has formulated the optimization problem at first place. The interested reader can refer to [139] for further details on the verification of the optimization problem.

In this chapter, first we describe the details of the unit testing practice we undertook, presented in Section 7.1, followed by the GUI testing method and scripts we designed to test the correctness of the functionality of the GUI, presented in Section 7.2. In Section 7.3, we presented the mutation testing performed on the optimization formulation script to further test the correctness of that script. Finally in Section 7.4, we discussed the lessons learned.

7.1 Unit Testing and NUnit Framework

Unit testing is a common standard practice in testing software projects as it is beneficial and applicable to all levels of programming languages (low level, middle level and high level) [163]. Unit testing refers to testing the individual units of the computer applications in order to make sure that system units are working as expected. When this activity is required to be automated, as in the testing of complex systems, unit testing frameworks are used. A unit testing framework, which is often a free open source software, in general provides the tester with a collection of key classes and functionalities, such as TestCase, to design, code and run unit tests [163]. TestCase class is used to implement the conditions and variables by which the system under test is being tested.

We have used the NUnit framework, which is a unit testing framework from the family of XUnit frameworks (such as JUnit and PyUnit), designed specifically to write and run unit tests for all .Net languages [164]. We created another project parallel to the project containing the source code of the system in Visual Studio to build our test suite for defining the test cases using NUnit framework. We generated a test suite containing a total of 151 test cases for the system. These test cases were generated using black-box testing method. By black-box testing here, we mean testing the functionality of the individual software units (by units here we mean methods in particular) against the requirements of the system, regardless of the source code. The expected values for the test cases are defined based on the expected functionality of the software features. It is worth mentioning that in this section we present the testing procedure we undertook for testing the software only, not the optimization module.

As performing exhaustive testing to cover all the possible combinations of input domain values is not feasible in most software applications, several testing techniques are devised to systematically reduce the number of test cases required to test a system. We employed "category partitioning" method, in order to generate our test cases. In category partitioning, the input domain for the method under test is divided into conceptually independent partitions and then a test value will be selected from each partition to generate a particular test case [161].

For example, assume that we want to test the getKMLPath method. This method gets a string value representing the path for a XML file as its argument and returns a string which is the path for the corresponding KML file. Instead of testing the method with all the possible input string values, we can generate 2 test cases to compare the method actual return values against the expected values defined by the method specifications. In this particular situation, input strings can be divided into two separate partitions; one the set of all the strings referring to a path where a XML file is saved and another set where either the string is not a valid path or it is referring to location where no XML file can be found. The expected return value for any nominal string taken from the first set is a string referring to a valid path where the corresponding KML file is stored. The expected return value for the nominal strings taken from the second set is a blank (according to the method specification decided in the implementation phase). All the other test cases are generated using the same method. A snapshot of the test cases generated for getKMLPath method in Visual Studio is shown in Figure 36.

After completion of the coding in each development iteration, we run the test suite and the test case failures (if any) were investigated for the root cause of the failure in order to remove the defects.

```
/// <summary>
///A test for getkmlPath
///</summary>
[TestMethod()]
public void getKMLPathTestNominalRightPath()
{
   mainController target = new mainController();
   string xmlpath11 = @"C:\Users\Roshanak\Dropbox\Project\Pipeline
Optimizer\bin\Release\data.xml";
   string expected = @"h:\Program Files\Microsoft Visual Studio
10.0\Common7\IDE\\Pembin_Pipeline.kml";
   string actual;
   actual = target.getkmlPath(xmlpath11);
   Assert.AreEqual(expected, actual);
}

/// <summary>
///A test for getkmlPath
///</summary>
[TestMethod()]
public void getKMLPathTestNominalWrongPath()
{
   mainController target = new mainController();
   string xmlpath11 = @"C:\Documents and Settings\rfarhood\My
Documents\Dropbox\Project\Pipeline Optimizer\bin\Release\data1.xml";
   string expected = "";
   string actual;
   actual = target.getkmlPath(xmlpath11);
   Assert.AreEqual(expected, actual);
}
```

Figure 36: Test cases generated for getkmlPath method

Table 23 summarizes the information related to the number of unit test cases generated for each of the classes of our application. As shown in the table, we generated 93 test cases for the methods of mainController class (methods are described in Table 21), 4 test cases for the optimizerController class and a total of 54 test cases of the model classes of the system. Breakdown of the number of the test cases generated for mainController class methods are tabulated in Table 24.

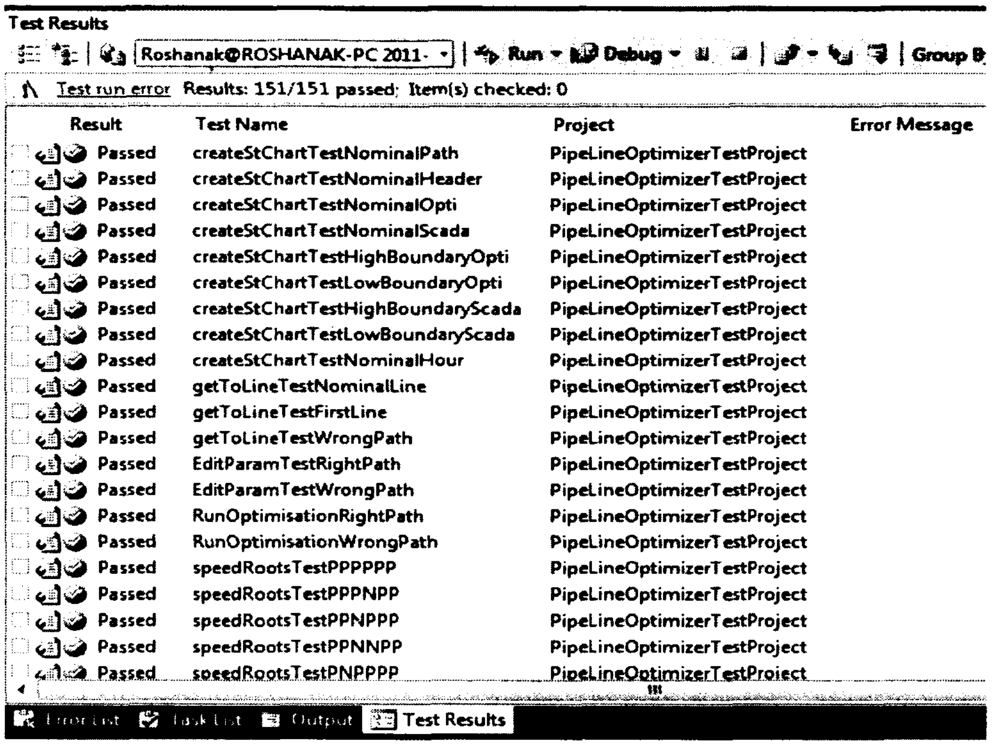
System classes	Class type	# of unit test cases
Pump	Model	40
PumpStation	Model	10
Network	Model	4
MainController	Controller	93
GEController	Controller	0
OptimizerController	Controller	4
GEWindow	View	0
FChart	View	0
Total		151

Table 23: Overview of the system classes and the number of the generated unit test cases using category partitioning approach

Method name	# of test cases
createOptChart	9
createSpeedChart	9
createStChart	9
getkmlPath	2
getToLine	3
loadCombo	3
readNetworkData	28
setOptimizationPath	2
speedRoots	16
getOptimisationPath	2
getxmlPath	2
setxmlPath	2
getImageC	2
setImageC	2
getOptCost	2
Total	93

Table 24: Number of test cases generated for methods of mainController class

The corresponding test run results, performed in Visual Studio is shown in Figure 37. As seen in the snapshot, all the 151 test cases passed and we had no test failures in this specific test run. Sample test cases can be found in Appendix C.



Test run error Results: 151/151 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
Passed	createStChartTestNominalPath	PipeLineOptimizerTestProject	
Passed	createStChartTestNominalHeader	PipeLineOptimizerTestProject	
Passed	createStChartTestNominalOpti	PipeLineOptimizerTestProject	
Passed	createStChartTestNominalScada	PipeLineOptimizerTestProject	
Passed	createStChartTestHighBoundaryOpti	PipeLineOptimizerTestProject	
Passed	createStChartTestLowBoundaryOpti	PipeLineOptimizerTestProject	
Passed	createStChartTestHighBoundaryScada	PipeLineOptimizerTestProject	
Passed	createStChartTestLowBoundaryScada	PipeLineOptimizerTestProject	
Passed	createStChartTestNominalHour	PipeLineOptimizerTestProject	
Passed	getToLineTestNominalLine	PipeLineOptimizerTestProject	
Passed	getToLineTestFirstLine	PipeLineOptimizerTestProject	
Passed	getToLineTestWrongPath	PipeLineOptimizerTestProject	
Passed	EditParamTestRightPath	PipeLineOptimizerTestProject	
Passed	EditParamTestWrongPath	PipeLineOptimizerTestProject	
Passed	RunOptimisationRightPath	PipeLineOptimizerTestProject	
Passed	RunOptimisationWrongPath	PipeLineOptimizerTestProject	
Passed	speedRootsTestPPPPPP	PipeLineOptimizerTestProject	
Passed	speedRootsTestPPPNPP	PipeLineOptimizerTestProject	
Passed	speedRootsTestPPNPPP	PipeLineOptimizerTestProject	
Passed	speedRootsTestPPNNPP	PipeLineOptimizerTestProject	
Passed	speedRootsTestPNPPPP	PipeLineOptimizerTestProject	

Figure 37: Snapshot taken in Visual Studio after running test methods of mainController class, showing all 151 test cases were passed in this run

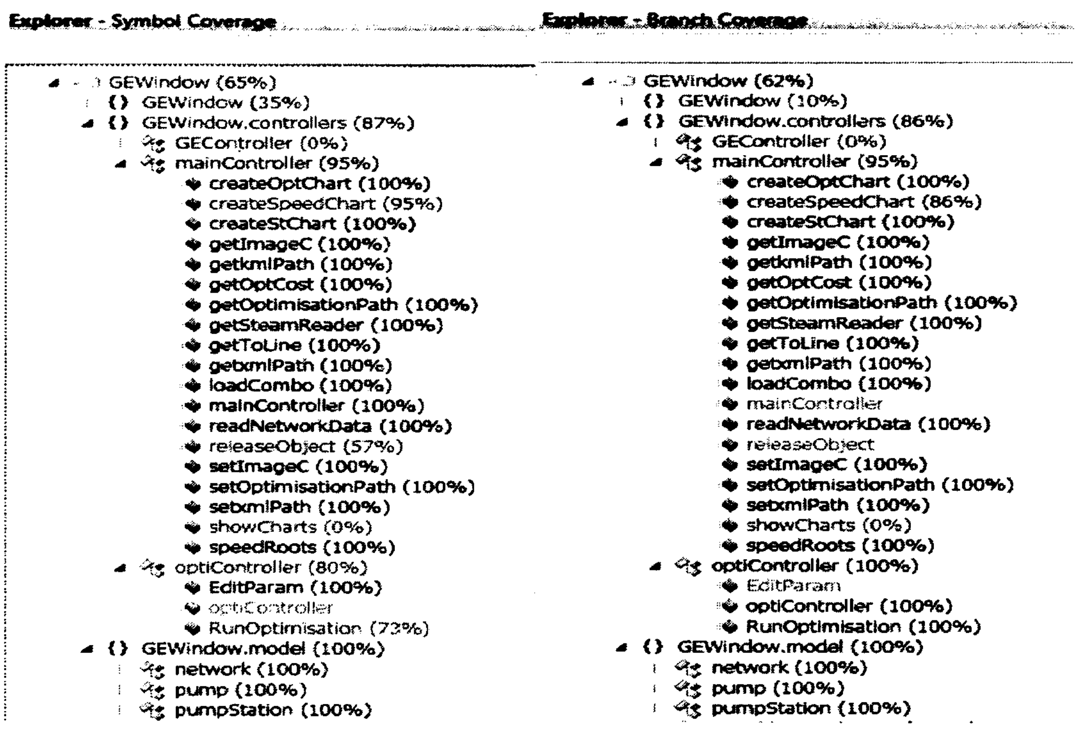
We did not generate test cases for the rest of the classes in the source code: We did not perform unit testing on the category of view classes, as we used windows-based standard .Net forms. We assume these .Net components are previously tested adequately (before deploying .NET framework) and in order to test the functionality embedded in the forms, we performed GUI testing. The methods used in GEController are the methods taken from Google Earth API (referenced from EARTHLib.dll), for managing the user interactions with Google Earth, such as zoon in, zoom out, drag, click to open the information box of a certain object shown on the map and resizing. We assume these methods were sufficiently tested by Google team while releasing Google Earth [165].

7.1.1 *Code Coverage*

One of the measures used in software testing is code coverage and it shows how adequately the source code is tested using the test suite. There exist different types of code coverage criteria. In this work, we measured the coverage based on symbol coverage and branch coverage criteria, as these two types of coverage were supported by the tool we used for test coverage analysis.

Symbol coverage, similar to line coverage, measures how many sequence points are covered by the test cases [166]. As we mentioned in Chapter 6, sequence points are used to highlight a spot in the IL code that corresponds to a certain location (usually a part of a statement) in the original code. Therefore, a statement can be broken down into several sections each of which is referred to by a sequence point. Branch coverage measures which decision outcomes in the source code are tested by the test suite.

The coverage was measured using NCover [166]. NCover is a .Net code coverage tool by which the users can investigate the parts of the code that are not yet covered by the test suite. The symbol and branch coverage scores calculated and reported by NCover are shown in Figure 38. As seen in the figure, using our test suite, we got 95% symbol and branch coverage for the mainController class and 100% symbol and branch coverage for the methods of model classes.



Explorer - Symbol Coverage

```
GEWindow (65%)
  {} GEWindow (35%)
  {} GEWindow.controllers (87%)
    GEController (0%)
    mainController (95%)
      createOptChart (100%)
      createSpeedChart (95%)
      createStChart (100%)
      getImageC (100%)
      getkmlPath (100%)
      getOptCost (100%)
      getOptimisationPath (100%)
      getSteamReader (100%)
      getToLine (100%)
      getxmlPath (100%)
      loadCombo (100%)
      mainController (100%)
      readNetworkData (100%)
      releaseObject (57%)
      setImageC (100%)
      setOptimisationPath (100%)
      setxmlPath (100%)
      showCharts (0%)
      speedRoots (100%)
    optiController (80%)
      EditParam (100%)
      optiController
      RunOptimisation (73%)
  {} GEWindow.model (100%)
    network (100%)
    pump (100%)
    pumpStation (100%)
```

Explorer - Branch Coverage

```
GEWindow (62%)
  {} GEWindow (10%)
  {} GEWindow.controllers (86%)
    GEController (0%)
    mainController (95%)
      createOptChart (100%)
      createSpeedChart (86%)
      createStChart (100%)
      getImageC (100%)
      getkmlPath (100%)
      getOptCost (100%)
      getOptimisationPath (100%)
      getSteamReader (100%)
      getToLine (100%)
      getxmlPath (100%)
      loadCombo (100%)
      mainController
      readNetworkData (100%)
      releaseObject
      setImageC (100%)
      setOptimisationPath (100%)
      setxmlPath (100%)
      showCharts (0%)
      speedRoots (100%)
    optiController (100%)
      EditParam
      optiController (100%)
      RunOptimisation (100%)
  {} GEWindow.model (100%)
    network (100%)
    pump (100%)
    pumpStation (100%)
```

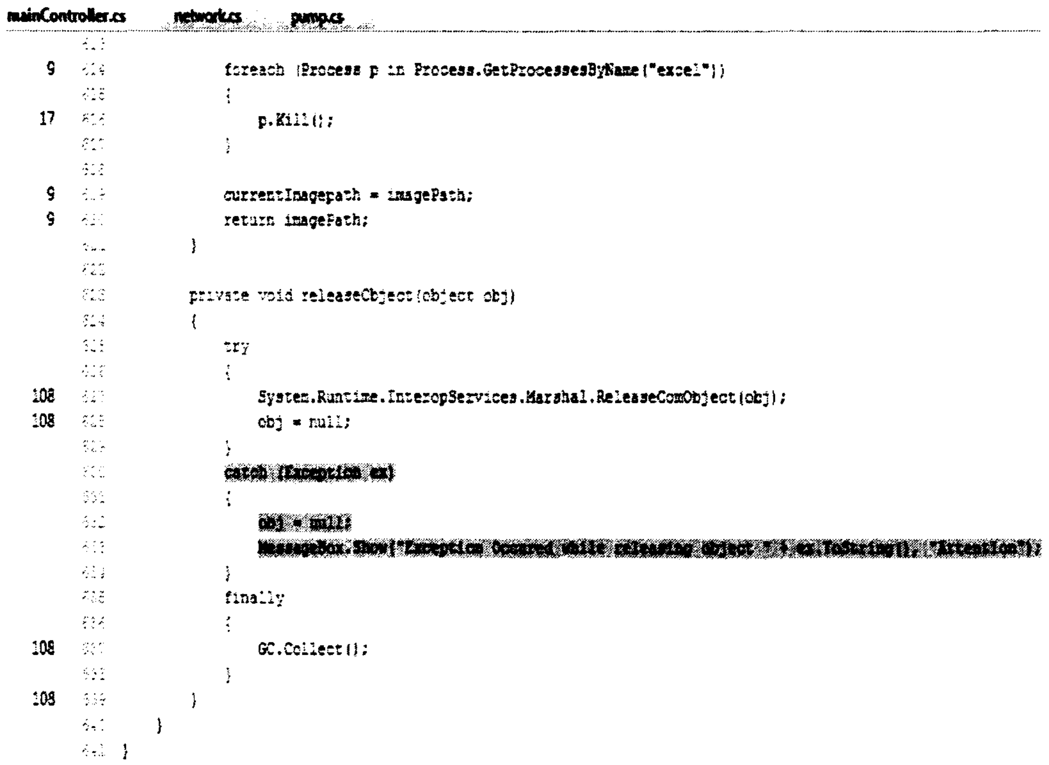
Figure 38: Symbol and branch coverage values taken from NCover, the elements which are not covered 100% by the test suite are color coded in red

A snapshot of the covered and uncovered code for releaseObject method in the mainController class by the test cases is shown in Figure 39. The uncovered code is highlighted in red.

7.1.2 *SUT and Test Suite Dependencies*

As software projects are always subject to being evolved or maintained, source code change is inevitable. Test impact analysis [167] is performed to analyze the code changes and to help the selection of those unit tests that are impacted by the source code change, which means it brings traceability into the testing practice.

Out test suite consists of several classes each of which includes the test cases for a certain method. We have depicted the relationships among the system artefacts and the test suite classes in a graph called Test Coverage Graph [168], as shown in Figure 40.



```
  9            foreach (Process p in Process.GetProcessesByName("excel"))
               {
 17                p.Kill();
               }

  9            currentImagepath = imagePath;
  9            return imagePath;
           }

           private void releaseObject(object obj)
           {
               try
               {
108                System.Runtime.InteropServices.Marshal.ReleaseComObject(obj);
108                obj = null;
               }
               catch (Exception ex)
               {
                   obj = null;
                   MessageBox.Show("Exception Occured while releasing object " + ex.ToString(), "Attention");
               }
               finally
               {
108                GC.Collect();
               }
108        }
       }
   }
```

Figure 39: Snapshot of the covered and uncovered code in NCover for releaseObject method in the mainController class

This mapping assisted us in performing our testing activity more efficiently, by demonstrating the dependencies among test classes and corresponding methods, which make the maintenance of the test suite more manageable, faster and easier whenever source code changed by introducing the traceability concept to the testing practice.

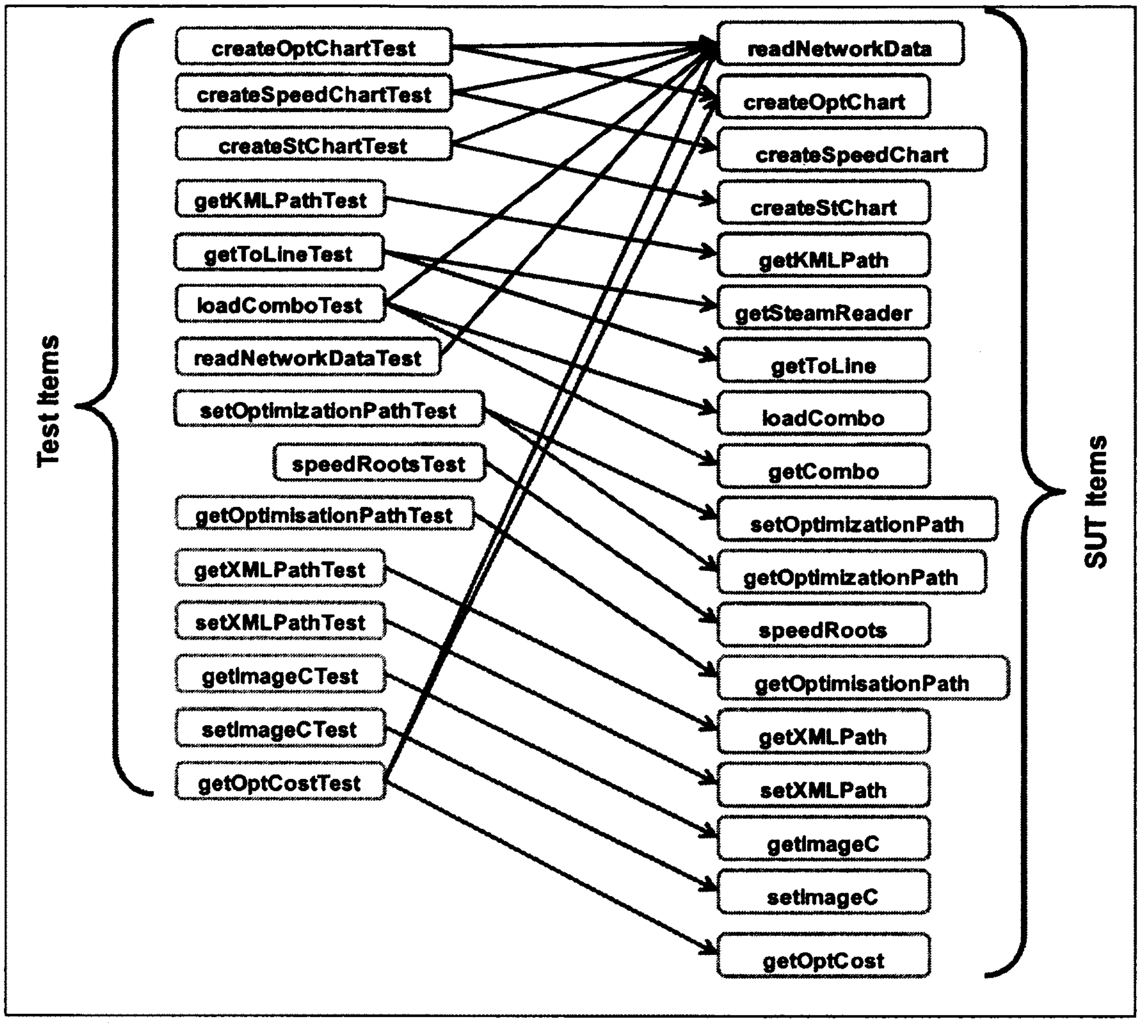


Figure 40: Test Coverage Graph for the test methods of mainController

7.2 GUI Testing

GUI testing (Graphical User Interface) is the process of testing the system's graphical user interface to assure it functions as expected. As a result, in practice GUI testing is often considered a major part of functional testing, which is testing of all features and functions of the system to ensure the requirements and specifications are all met [161, 169]. In general to generate adequate set of GUI test cases, all the functionality of the system needs to be covered so that the test suite fully exercises all the possible events generated using the GUI [170, 171]. As our system consists of GUI components to offer the system functionality to the operators, in this section we describe our practice of testing the system GUI.

7.2.1 *Event-Flow Graph*

In order to simplify testing the system functionality through its GUI, we extracted and investigated high level event-flow graph [169] of the system. GUI test cases are then generated by traversing this graph in order to test the sequence of the events created by the user while interacting with the system [169, 171]. The high level event-flow graph of the system, depicted as the interaction overview diagram (a type of UML activity diagram), is shown in Figure 41.

As seen in the figure, the operator can start working with the system by loading a new pipeline to the system. This gives the operator the possibility of browsing pipeline elements in Google Earth. Then the operator can open the optimization formulation file to edit the target parameter(s) and then run the optimization solver. This often is followed by viewing the optimal operation parameter values and comparing them with SCADA data using comparative charts.

7.2.2 *GUI Events and Widgets*

GUI events are created when the user interact with the system using GUI components [169]. These components are called widgets in this context, such as menus, buttons, text boxes and combo boxes [171]. Different types of possible events generated in our system's GUI are summarized in Table 25.

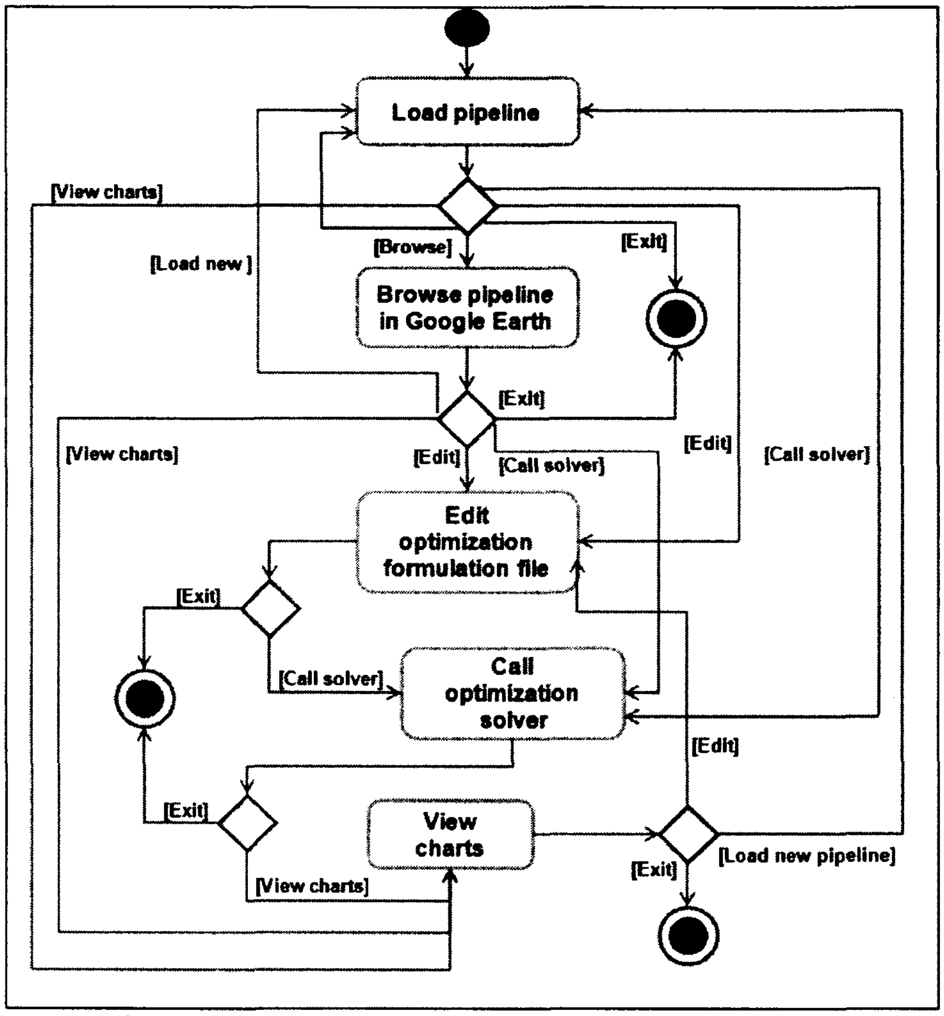


Figure 41: Interaction overview diagram of the system

Event type	Corresponding Widget	Corresponding user action
Mouse left click	Main menu	When the user left clicks on the main menu to select one of the menu options
Mouse left click	Submenu	When the user left clicks to select one of the sub menu options
Mouse left click	Button	When the user clicks on a button
Text change	Combo Box	When the user changes the selected option of the combo box
Selected change	Radio button	When the user changes the status of a radio button from selected to unselected and vice versa

Table 25: System GUI events, their corresponding widget and user actions

7.2.3 *Event Sequences*

In order to accomplish a target task in the system, as shown in the event-flow graph, the user is required to pass through several steps while interacting with the system, which results in the creation of a sequence of the events. These events are summarized in Table 26.

Target	Event sequences generated by the user
Load new pipeline	Click main menu "File", click "Load new pipeline", clicks to choose the target XML data file, click "Ok"
Browse the pipeline	Zoom in, zoom out, drag, click on pump stations to open the information window, click to close the information window
Edit optimization formulation file	Click main menu "Tools", click "optimization", click "Edit parameters", save file, close
Call the solver	Click main menu "Tools", click "optimization", click "Run"
View charts	Click main menu "Tools", click "View optimization charts", choose chart type by clicking "chart type" radio button, choose station name by clicking on the corresponding combo box, close "view charts" window
Close the application	Click main menu "File", click "Exit"

Table 26: Summary of the events generated by the user interacting with the system

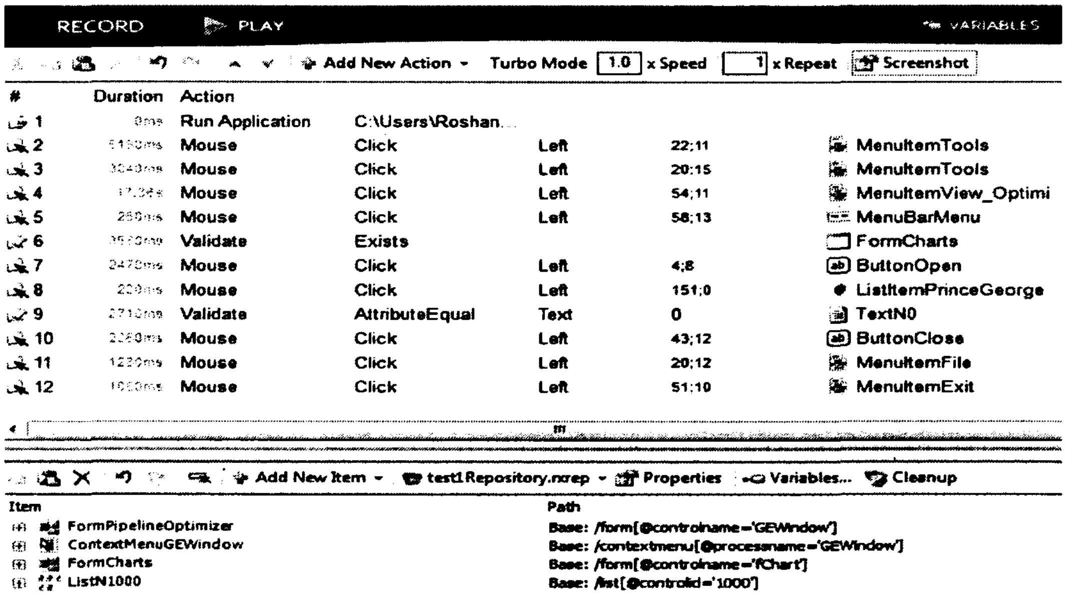
7.2.4 *GUI Testing Tool*

There exist several commercial tools for GUI testing, such as IBM rational functional Tester by IBM [172] and Visual Studio 2010 Premium edition [173], each of which offer a set of features to generate test cases and manage the testing activity for large-scale GUI-based software applications. There also exist open source software such as NUnitForms [174] which is an NUnit extension for unit and acceptance testing of windows forms applications and GUITAR [175] which is usually used for Java-based applications.

We used Ranorex Studio [176] to perform GUI testing, as this tool was easy to setup and provided us with the required features to generate our test scripts. Ranorex Studio supports test implementation and GUI script record and play back facility for the applications developed in C#. We employed "record/play back" [177] method. In this method the test scripts are recorded using the recorder utility provided by the test tool and then the scripts will be played back to check the correctness of system functionality.

7.2.5 *GUI test Cases (Test Scripts)*

To generate GUI test scripts, the event-flow graph of the system was traversed manually to extract possible paths, representing the scenarios when the user interacts with the system. Using the breakdown of the events presented in Table 26, we recorded 23 GUI test scripts, primarily based on the event-interaction coverage criterion that requires all the edges of the event-flow graph to be covered by at least one test case [178]. Additional to getting 100% event-interaction coverage score, we added some test scripts to cover some common scenarios that an operator may go through while working with the system. These scripts are listed in Table 27.



#	Duration	Action				
1	0ms	Run Application	C:\Users\Roshan...			
2	5150ms	Mouse	Click	Left	22;11	MenuItemTools
3	3040ms	Mouse	Click	Left	20;15	MenuItemTools
4	17.34s	Mouse	Click	Left	54;11	MenuItemView_Optimi
5	250ms	Mouse	Click	Left	58;13	MenuBarMenu
6	3520ms	Validate	Exists			FormCharts
7	2470ms	Mouse	Click	Left	4;8	ButtonOpen
8	200ms	Mouse	Click	Left	151;0	ListItemPrinceGeorge
9	2710ms	Validate	AttributeEqual	Text	0	TextN0
10	2260ms	Mouse	Click	Left	43;12	ButtonClose
11	1220ms	Mouse	Click	Left	20;12	MenuItemFile
12	1060ms	Mouse	Click	Left	51;10	MenuItemExit

Item	Path
FormPipelineOptimizer	Base: /form[@controlname='GEWindow']
ContextMenuGEWindow	Base: /contextmenu[@processname='GEWindow']
FormCharts	Base: /form[@controlname='fChart']
ListN1000	Base: /list[@controlid='1000']

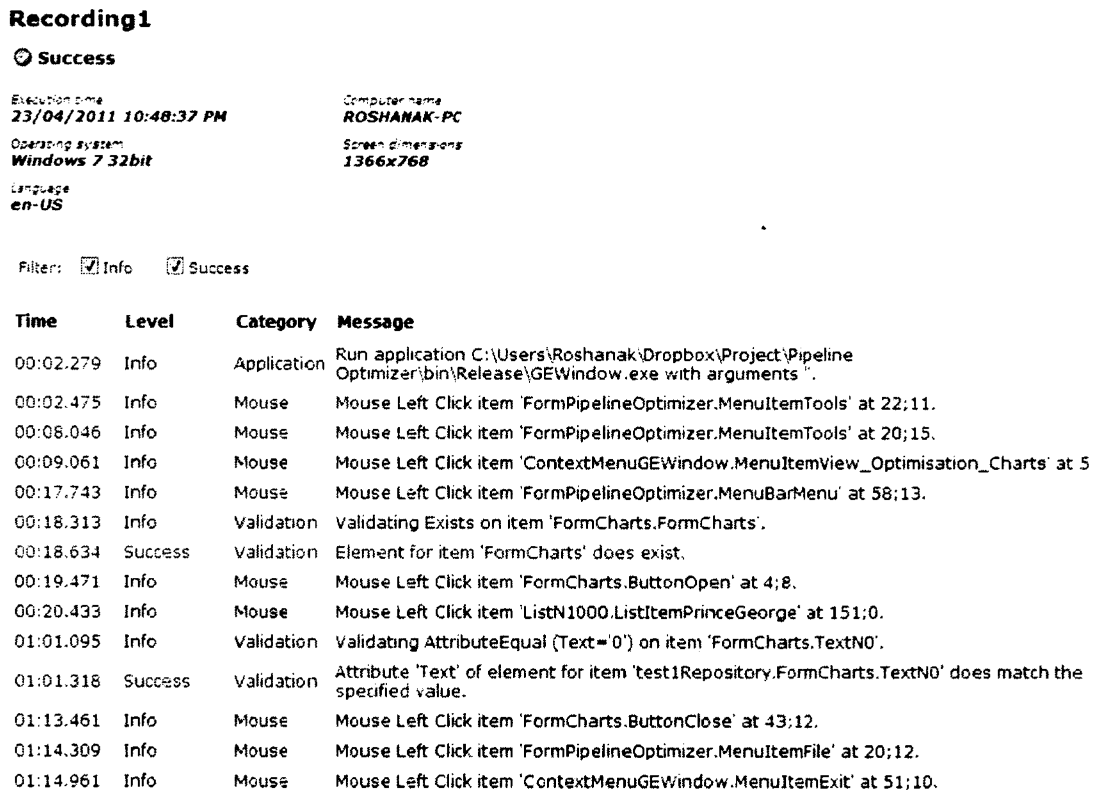
Figure 42: Snapshot of test script taken from Ranorex Studio environment showing different GUI actions included in the script

A snapshot of a test script (script number 4 in Table 27) recorded in Ranorex Studio testing environment is shown in Figure 42.

Test Script #	Paths
1	Load pipeline-Close
2	Load pipeline-Load pipeline-Close
3	Load pipeline-Browse pipeline- Close
4	Load pipeline-Browse pipeline-View charts -Close
5	Load pipeline-View charts- Close
6	Load pipeline-Browse pipeline-Edit formulation file- Close
7	Load pipeline-Browse pipeline-Edit formulation file- Call solver- Close
8	Load pipeline-Browse pipeline-Edit formulation file- Call solver-View charts-Close
9	Load pipeline-Browse pipeline-Edit formulation file-Call solver-View charts-Load new pipeline- Close
10	Load pipeline-Browse pipeline-Edit formulation file-Call solver-View charts-Load new pipeline- Browse pipeline- Close
11	Load pipeline-Run solver- Close
12	Load pipeline-Run solver-View charts- Close
13	Load pipeline- Edit formulation file-Run solver- Close
14	Load pipeline- Edit formulation file-Run solver-View charts- Close
15	Load pipeline-View charts- Edit formulation file-Run solver- Close
16	Load pipeline-View charts- Edit formulation file-Run solver-View charts- Close
17	Load pipeline-Run solver-View charts-Edit formulation file-Run solver- Close
18	Load pipeline-Run solver-View charts-Edit formulation file-Run solver-View charts-Close
19	Load pipeline-Run solver-View charts-Edit formulation file-Run solver-View charts- Edit formulation file-Run solver-View charts -Close
20	Load pipeline-Run solver-View charts- Edit formulation file-Run solver-View charts- Edit formulation file-Run solver-View charts-Load new pipeline-Run solver-View charts-Close
21	Load pipeline-Browse pipeline- Run solver –Close
22	Load pipeline-Browse pipeline- View charts –Close
23	Load pipeline-Browse pipeline- Load new pipeline- Close

Table 27: Different paths used to record GUI scripts

As seen in the snapshot, the script recorded corresponds to a scenario where in the application, user chooses to open "view optimization chart" window (line number 4 as shown in the snapshot) and in that window "S3" station is selected and the value of total optimization cost is validated (line number 9 as shown in the snapshot), the close button of "view optimization chart" window is clicked and then the user chooses to exit the application. In the "play back" mode in Ranorex Studio, the recording can be played back and the validations as well as the sequence of the events can be checked to assure the correct system functionality. A snapshot showing the success of the playback corresponding to above scenario is shown in Figure 43.



Recording1

Success

Execution time	Computer name
23/04/2011 10:48:37 PM	*ROSHANAK-PC*
Operating system	Screen dimensions
Windows 7 32bit	*1366x768*
Language	
en-US	

Filter: [✓ Info] [✓ Success]

Time	Level	Category	Message
00:02.279	Info	Application	Run application C:\Users\Roshanak\Dropbox\Project\Pipeline Optimizer\bin\Release\GEWindow.exe with arguments ''.
00:02.475	Info	Mouse	Mouse Left Click item 'FormPipelineOptimizer.MenuItemTools' at 22;11.
00:08.046	Info	Mouse	Mouse Left Click item 'FormPipelineOptimizer.MenuItemTools' at 20;15.
00:09.061	Info	Mouse	Mouse Left Click item 'ContextMenuGEWindow.MenuItemView_Optimisation_Charts' at 5
00:17.743	Info	Mouse	Mouse Left Click item 'FormPipelineOptimizer.MenuBarMenu' at 58;13.
00:18.313	Info	Validation	Validating Exists on item 'FormCharts.FormCharts'.
00:18.634	Success	Validation	Element for item 'FormCharts' does exist.
00:19.471	Info	Mouse	Mouse Left Click item 'FormCharts.ButtonOpen' at 4;8.
00:20.433	Info	Mouse	Mouse Left Click item 'ListN1000.ListItemPrinceGeorge' at 151;0.
01:01.095	Info	Validation	Validating AttributeEqual (Text='0') on item 'FormCharts.TextN0'.
01:01.318	Success	Validation	Attribute 'Text' of element for item 'test1Repository.FormCharts.TextN0' does match the specified value.
01:13.461	Info	Mouse	Mouse Left Click item 'FormCharts.ButtonClose' at 43;12.
01:14.309	Info	Mouse	Mouse Left Click item 'FormPipelineOptimizer.MenuItemFile' at 20;12.
01:14.961	Info	Mouse	Mouse Left Click item 'ContextMenuGEWindow.MenuItemExit' at 51;10.

Figure 43: Snapshot taken after playing back the test script shown in Figure 42, showing the success of the validations included in the script

7.3 Mutation Testing on Optimization Formulation Script

Mutation testing is a testing technique which is conducted in order to assess the test suite adequacy for detecting software defects and also to improve the code coverage by the test cases [161]. During mutation testing, small syntactic changes, such as arithmetic or logical operator changes are made to the source code. As a result, a set of similar faulty programs called mutants are created. Then we run the test suite on these faulty versions, and if any test case fails while testing a mutant, the mutant is said to be killed. If the existing test cases in a test suite can not kill the mutants, the test suite is not adequate.

```
! S1 station with two identical pumps H_S11, H_S12 ;
@FOR(Station(s) | s #EQ# 1:
@BIN(B_S11(t)) ;
@BIN(B_S12(t)) ;
@BIN(Te_S1(t)) ;
H_S1_Suc(t) = 1238000/ro/g ;
H_S1_Disch(t) = H_S1_Suc(t) + H_S11(t) + H_S12(t) ;
P_S11(t) = P_Q_S11 * (Q_T(t) - Q_S11(t)) + P_H_S11 * H_S11(t) +
P_Icpt_S11 * B_S11(t) ;
P_S12(t) = P_Q_S12 * (Q_T(t) - Q_S12(t)) + P_H_S12 * H_S12(t) +
P_Icpt_S12 * B_S12(t) ;
H_S11(t) < B_S11(t) * H_Max ;
H_S11(t) > B_S11(t) * H_min ;
H_S12(t) < B_S12(t) * H_Max ;
H_S12(t) > B_S12(t) * H_min ;
Q_S11(t) > (1-B_S11(t)) * Q_min ;
Q_S11(t) < (1-B_S11(t)) * Q_Max ;
Q_S12(t) > (1-B_S12(t)) * Q_min ;
Q_S12(t) < (1-B_S12(t)) * Q_Max ;
C_S11(t) = P_L_S1(t) * Rate_L_S1(t) + P_H_S1(t) * Rate_H_S1(t) ;
P_L_S1(t) + P_H_S1(t) = P_S11(t) + P_S12(t) ;
P_L_S1(t) < P_Thresh_S1 * Te_S1(t) ;
P_H_S1(t) > P_Thresh_S1 * (1-Te_S1(t)) ;
P_H_S1(t) <= 100000 * (1-Te_S1(t)) ;
H_1000(t) = H_S1_Disch(t) + (HS_S1 - HS_1000) - V_1000(t) - (a* Q_T(t)
+ b) * l_S1_1000 ;
H_1001(t) = H_1000(t) + (HS_1000 - HS_1001) - V_1001(t) - (a * Q_T(t)
+ b) * l_1000_1001 ;
H_Sunset(t) = H_1001(t) + (HS_1001 - HS_Sunset) - (a * Q_T(t) + b) *
l_1001_Sunset ;
H_1049(t) = H_Sunset(t) + (HS_Sunset - HS_1049) - V_1049(t) - (a *
(Q_T(t)+Q_SP(t)) + b) * l_Sunset_1049 ;
H_1051(t) = H_1049(t) + (HS_1049 - HS_1051) - V_1051(t) - (a *
(Q_T(t)+Q_SP(t)) + b) * l_1049_1051 ;
H_S3_Suc(t) = H_1051(t) + (HS_1051 - HS_S3) - (a * (Q_T(t)+Q_SP(t)) +
b) * l_1051_S3 ;
);
```

Figure 44: Sample part of the optimization formulation file [139]

As we could achieve high code coverage with our test suite, as described in section 7.1.1, we did not perform mutation testing on the application code. We applied mutation testing on the optimization script, besides the sensitivity analysis done by another member of our research group, in order to add more to the verification of the optimization formulation file. The formulation file format is similar to scripting languages, such as SQL. A piece of the optimization file is shown in Figure 44. Here in order for a mutant to be killed, we consider two cases: (1) if the optimization solution is not feasible, meaning that if the solver cannot find the global optimum, and (2) if the optimal values achieved from running the faulty script differ from their values achieved from running the original script.

In order to create mutants, we considered logical and arithmetic changes as well as random variable name changes and statement deletion. The changes are applied manually. The mutation testing is summarized in Table 28.

Mutation Operator	Number of Mutants	Description
Change < to >	27	All killed
Change > to <	32	All killed
Change + to -	20	All killed
Change * to /	20	All killed
Change <= to >=	5	All killed
Variable name change	20	All killed
Statement deletion (randomly selected)	10	All killed

Table 28: Mutation testing summary

As seen in the table, we have 27 occurrence of "<" and 32 occurrence of ">". We had over 130 occurrences of + and * operators, from which we randomly choose 20 operators to create faulty scripts. Also we performed a group of variable name changes, where the name of a variable was replaced with another variable. Some cases of randomly statement deletions were also considered to create mutants. Then we run the optimization engine to solve the faulty versions and we investigated the solver output to detect the changes in achieved optimal values or any infeasible states. As seen in the table all of our mutants were killed, which means we either achieved different optimal values compared to original optimal values after running faulty versions or the optimization solution was infeasible. This can be related to the high sensitivity incorporated with the nature of the pipeline problem formulation.

7.4 **Discussion**

According to the SLR performed as a part of this thesis, testing SES remains a great challenge for the practitioners and the complexity of verification and validation of SES is still an open issue. To elaborate our experience regarding the first research question of the case study conducted, we had to deal with the validation of the scientific part of the work (as called "scientific validation" by Hook and Kelly [75]) as well as the correctness of the software developed to utilise that scientific core (referred to as "code scrutinization" in [75]).

The first challenge is the result of not having certain oracles, as mentioned before, and the experience of testing the "engineering core" of our system confirms that in response to H1.3 (third hypothesis of the case study first research question), as no "expected optimal solution" were available upfront, so that the "actual optimal solutions" achieved from the developed optimization module could be tested against them.

To address the first challenge in response to H2.4 (forth hypothesis of the case study second research question), as reported in the related section in [139], the sensitivity analysis of the optimization formulation file was undertaken. Also the optimization problem was solved by another technique (i.e. genetic algorithm) so that the results of two methods can then be compared. If two methods generate similar results, this can add to the validity of the optimal results.

In order to address the second challenge, as described in this chapter, standard testing activities such as unit testing and GUI testing were conducted on the system, in order to assure correctness of the system's functionality.

We also brought the best practices taken from the results of the SLR into action; the study reported in [51] suggested having a test plan to go through the testing activity more efficiently, we had our test plan to systematically develop the required testing strategy as described in this chapter, which was integrated into our iterative development practice. The iterative style of the development also leads into running the test cases often as suggested in [88] to ensure the correctness of the system in each iteration.

Use of MVC architectural pattern as mentioned in Chapter 5 to introduce testability was also beneficial, as it provides the possibility of testing the functionality of the software independent from user interface. On the other side, further changes in the interface, which may happen as the result of customizing the interface for new clients, will not impact the core functionality of the system. This makes further testing and maintenance easier.

7.5 **Chapter Summary**

In this chapter, the testing methods and practice undertook in order to detect system bugs and assuring the system correct functionality was discussed. We employed black box unit testing, to test the methods of mainController class actual output against their expected output based on the function specifications. As a result, the source code coverage was also reported. Also in order for testing the functionality of the system through its GUI we used record and play back GUI testing method. We applied mutation testing on the optimization formulation file, in order to fortify the sensitivity analysis done by another member of our group. The summary of the lessons learned from testing our system along with the lessons learned and best practices reported by other researchers as applied to our case study, were presented.

Next Chapter will discuss several usage scenarios of the system. Some important features and the commercialization of the system will also be discussed.

Chapter Eight: **Operation and Usage**

Software systems are typically developed to support the users in performing their tasks more accurately and efficiently. Our pipeline operation system was primarily planned to play a decision support system role in making important decision in the pipeline operation.

In order to achieve this goal, we required to identify the scenarios for which the system can potentially be beneficial. Following a Behavior-Driven development approach [179], we first tried to understand in what situations the system is expected to be utilized by our industrial partner, so that it can best serve their business needs. Then the focus of the development, especially implementation of the features, will be on providing the system with the features which are beneficial to the users in the identified usage scenarios.

In Behaviour-Driven development, in order to identify the important scenarios of the system usage, critical questions such as "What is the most important thing the system should do?" or "Without using the system, where and what would be the biggest impact?" are posed [179]. Investigating the answers to such questions brings the insight on the importance of the system and the identification of the situations where not having the system may cause difficulties and challenges which can not be easily tackled.

As mentioned before, this system is developed with the objective of being used in optimizing the pipeline operation. Through our meetings with our industrial partner, we could identify every day concerns of the pipeline operators as well as the company manager. These concerns mostly were related to the power consumption of the pump stations and the operational speed of the variable speed pumps. We could address these concerns by designing and integrating the visualization of the power consumptions for each pump station and the pipeline network as a whole, as well as calculating and demonstrating the pump speeds within a 24 hour operation period. The details of a group of real world scenarios in which these features were found beneficial to the company will be presented in this Chapter.

In this Chapter, first we introduce the system's main features, which make it a decision support system by discussing several usage scenarios in Section 8.1. Other +features of the system, such as visualizing the pump speed optimal values and loading a new pipeline are then briefly presented in Section 8.2 and Section 8.3.

In the figures in this chapter, the values and the station names (vertical and horizontal axis labels) were made hidden intentionally to respect the confidentiality of the information belonging to our industrial partner.

8.1 **Usage Scenarios**

In this section we present three different situations where the use of the optimization software is studied in order to help a pipeline operator in making decisions:

- Scenario 1: The impact of the delivery volume changes on the total power cost, using the optimization charts,
- Scenario 2: The impact of replacing an existing pump with a new pump on the total power cost,
- Scenario 3: The impact of changes in power rates and thresholds on the total power cost.

These scenarios are designed to show that the developed application can be used as an effective decision support system for the company operators or manager when required to make important financial, practical and contractual decisions. This is done by providing them with an effective visualization to demonstrate how changing different factors may impact the total power cost.

In each of the scenarios described below, first the user should open the optimization dialogue window, open the optimization file there, which consequently opens the Lindo editing environment and changes the corresponding parameters in the optimization file. Then Lindo the optimization file should be run once and based on the new produced results the new optimization charts are generated and shown. The decision can be made by comparing the resulting charts. All the charts shown below are snapshots taken from the "charts" window of the application.

8.1.1 *Scenario 1: The impact of the delivery volume changes on the total power cost, using the optimization charts*

In this scenario, the delivery volume contract is changed and the impact of the changes is studied on the total power cost of the stations.

For example, let us assume the primary amount of volume contract is equal to 3,650 cubic meters based on the data taken from the company; the power consumption in a period of working 24 hours in each station as well as the total amount of power consumption is shown in Figure 45. The change in the power cost after decreasing the volume to 3,000 cubic meters is shown in Figure 46.

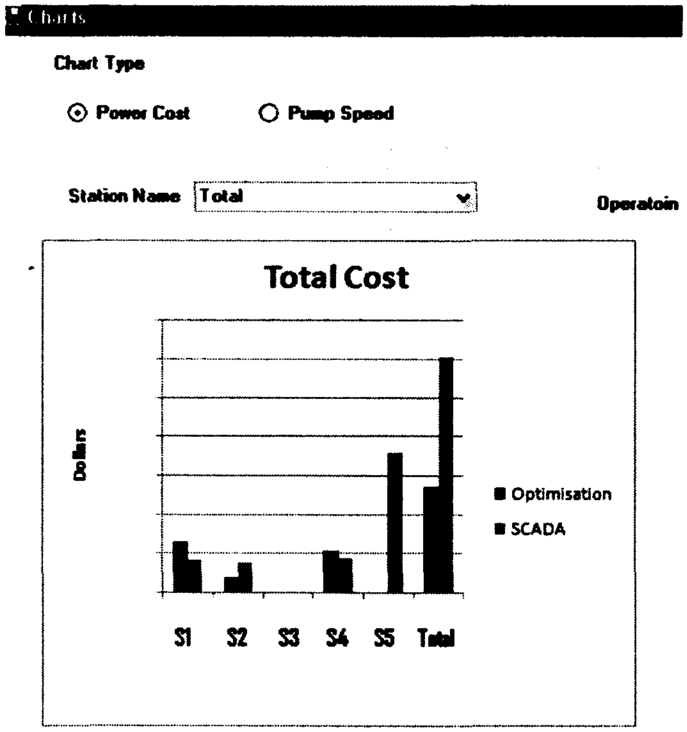


Chart Type: [✓ Power Cost] Pump Speed

Station Name: Total

Figure 45: Total cost, default case

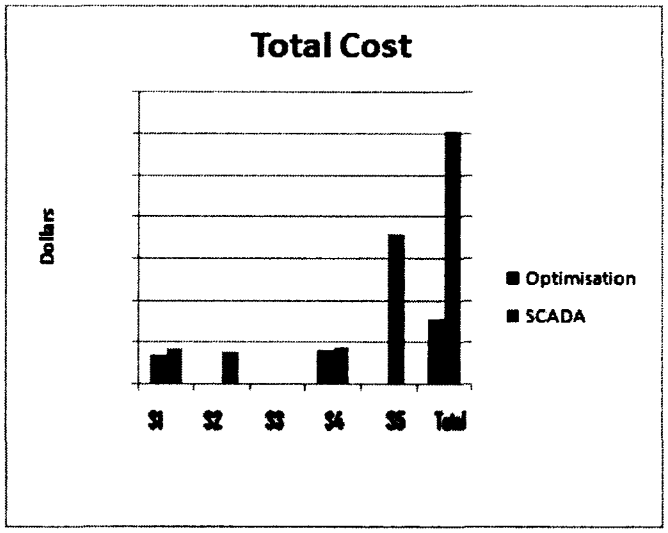


Figure 46: Total cost after decreasing volume

The change in power cost after increasing the volume to 3800 cubic meters is shown in Figure 47.

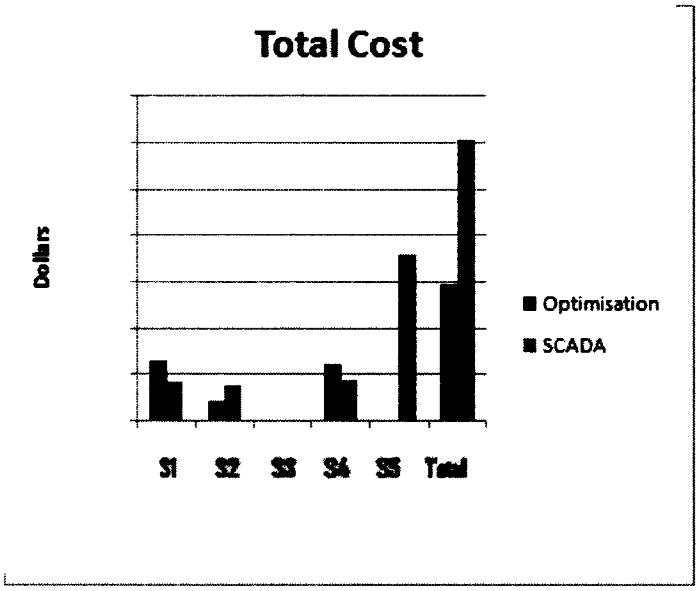
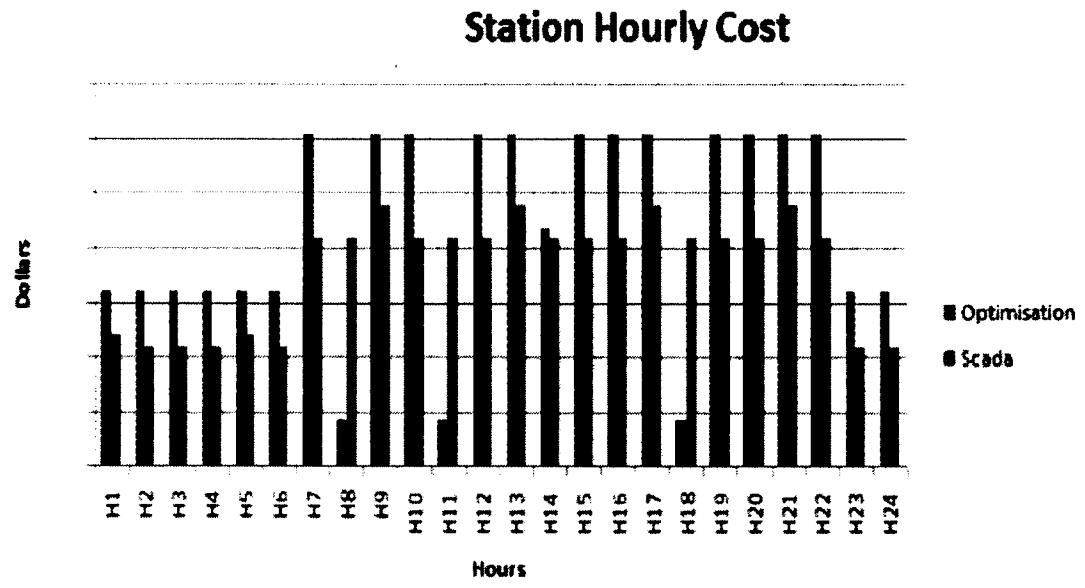


Figure 47: Total cost after increasing volume

As we have shown in the sample volumes above, the changes in volume will result in proportional changes in power cost. It is worth noting here that there exists a limited maximum allowable amount of volume in the pipeline under study which is determined based on the length, diameter and other hydraulic characteristics of each pipeline segment. If the volume amount is set to some values higher than this limit, no optimal solution will be found. Please refer to [139] for more details.

8.1.2 *Scenario 2: The impact of replacing an existing pump with a new pump on the total power cost*

This comparison can be used in a situation where the user wants to investigate whether it is a cost-effective decision to replace an old pump with a new one or not.



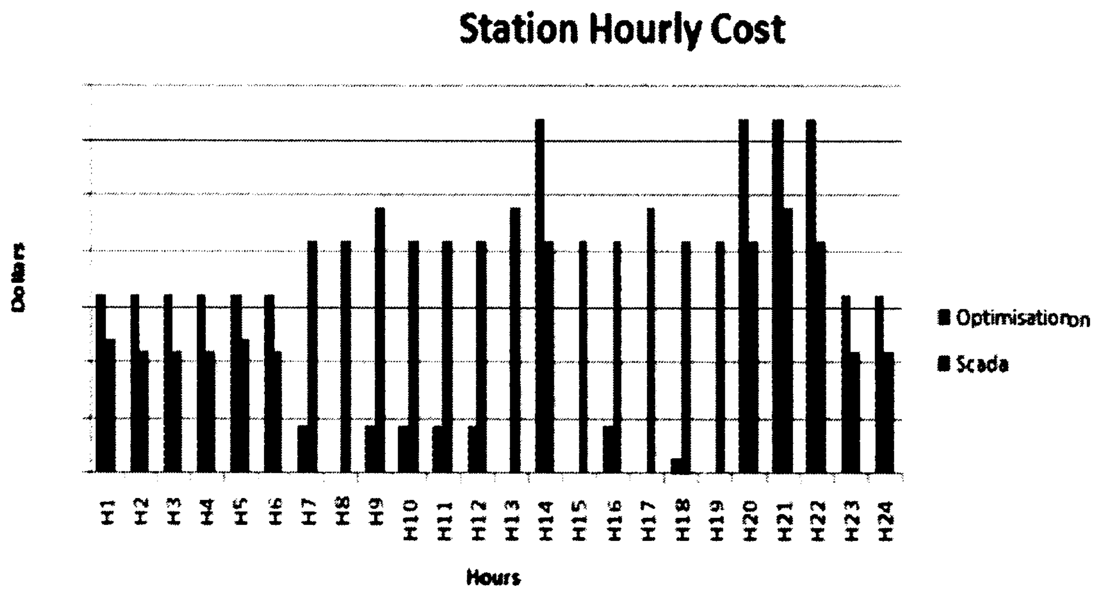
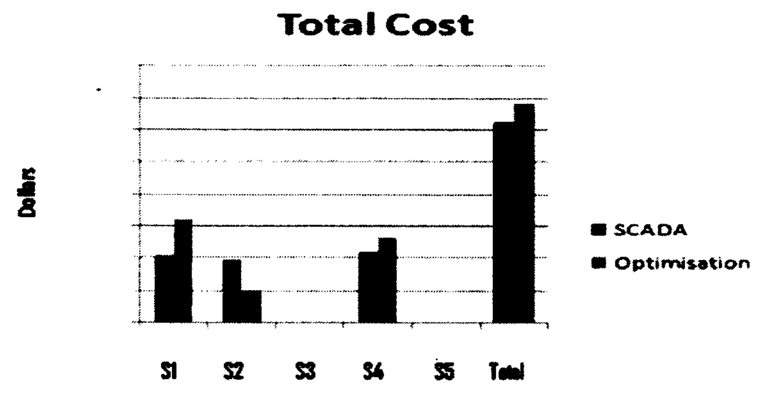


Figure 48: The impact of changing first pump in S4 station on power cost (top) before, (bottom) after replacing the pump with a pump similar to S2 pump station



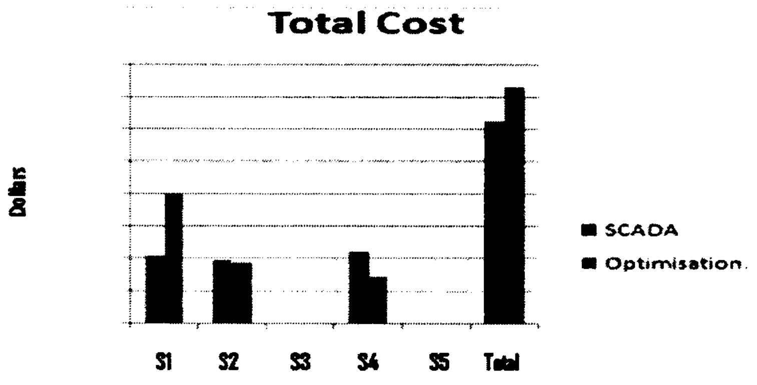
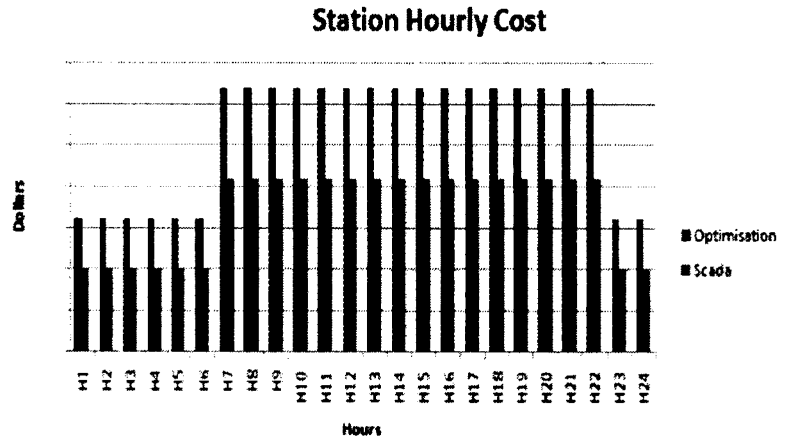


Figure 49: The impact of changing first pump in S4 station on other stations power cost and total power cost (top) before, (bottom) after replacing the pump with a pump similar to S2 pump station

As shown in Figure 48 and Figure 49, replacing the first pump in S4 station with another pump similar to that of S2 station will result in an increase in total pipeline power cost as well as an increase to power cost of S1 and S2 stations while we notice a decrease in the power cost of the S4 station itself. Interested reader can refer to [139] for more information about the details of pump parameters and how they are calculated and being used in the optimization objective function.

8.1.3 *Scenario 3: The impact of changes in power rates and thresholds on the total power cost*

Sometimes changes in power rates or power rate thresholds may result in dramatic changes in the total resulting power cost. By comparing the final effect of such changes in different situations, the user can decide better about the rates while negotiating for a new power contract. In the situation shown in Figure 50, the amount of power cost increase after doubling power rates for S1 station is demonstrated for different hours during the day.



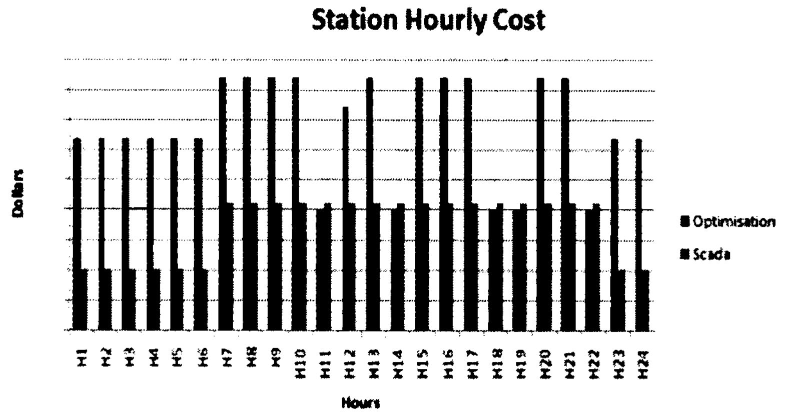
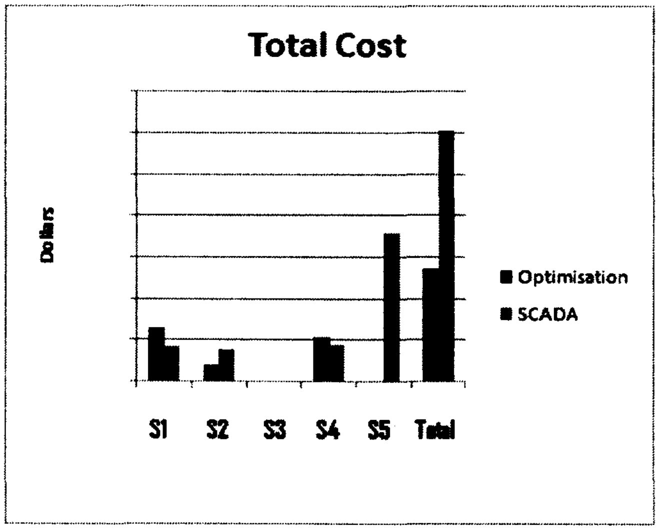


Figure 50: S1 station power cost (top) with default power rates, (bottom) after doubling power rates

The impact of this increase on the total power cost is demonstrated in Figure 51. As shown in the figure, the increase not only affect the total power cost of the S1 station, but also it resulted in an increase in total power cost of the neighbour station, S2, which resulted in the total power cost of the whole pipeline.



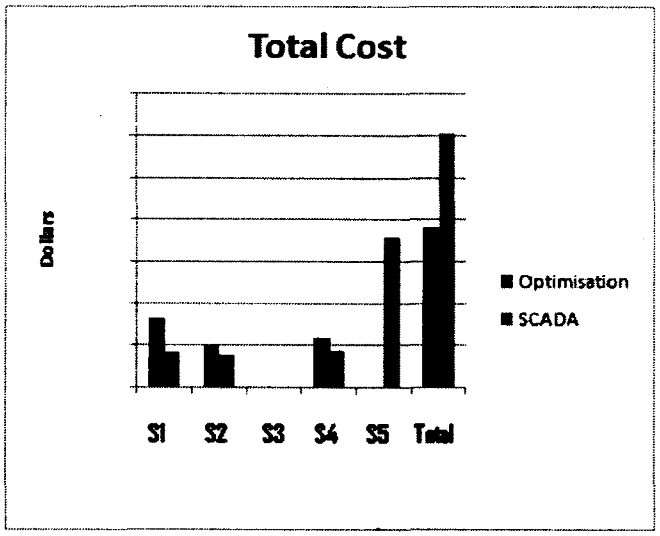


Figure 51: Impact of total power cost (top) before, (bottom) after doubling S1 station power rates

8.2 Speed Charts

As mentioned before, the pipeline operation optimization problem aims at finding the optimal pipeline setting under which the pipeline is operated at minimum cost. After this setting is found, the operator applies the proposed setting to the pipeline. This is achieved mainly by either turning on/off a certain pump or running the variable speed pumps with a target speed at certain hours.

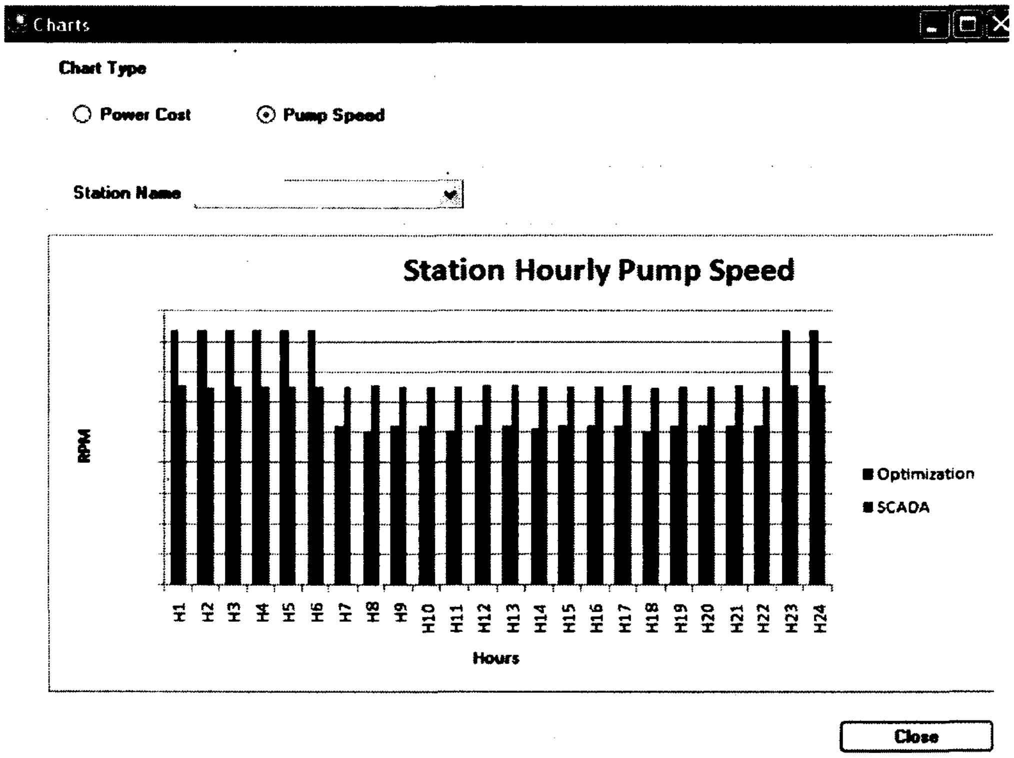


Figure 52: Sample speed chart

As the optimization solver does not directly provide the optimal operation values for pump speeds, another student of our research group, extracted the formula by which the pump speed is calculated using the optimal values found for pump head, flow rate and the coefficients of the pump. Employing that formula, our system finds the optimal speed and provides the operator with the target operating pump speed in different hours throughout the period optimization is performed. These values can be viewed in the charts form, by selecting the chart type as "Pump Speed". A snapshot of the optimal speed values calculated for a pump station is shown in Figure 52. These values are shown against the data taken from SCADA system.

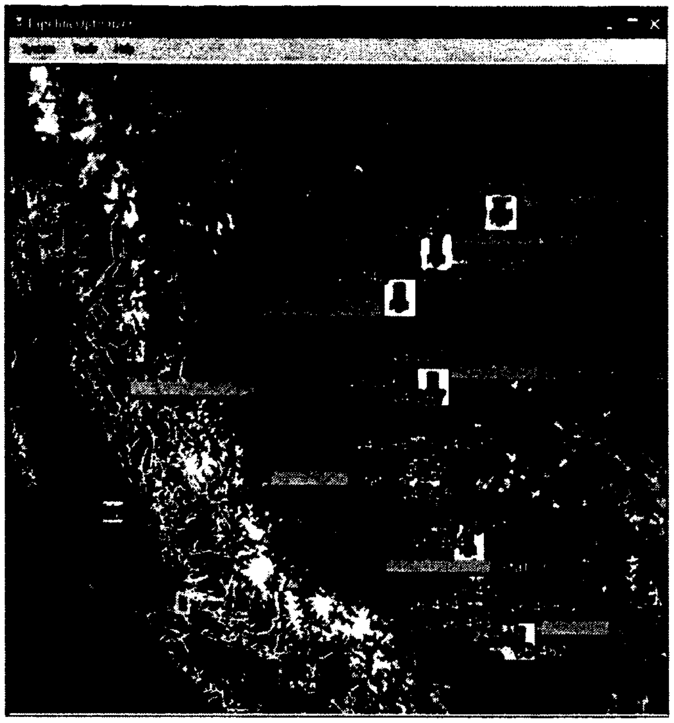
8.3 **Loading a New Pipeline**

One of the features provided by our system is that the user can load different pipelines to the system. As a result, all of the corresponding parameters and files are updated and the new schematic pipeline can be browsed in Google Earth.

Besides the pipeline operated by our industrial partner, we designed a sample pipeline in order for presenting the scenario of loading a new pipeline in the system. This test pipeline includes 3 variable speed pumps as shown in Figure 53.

The system updates the corresponding parameters based on the values provided in the XML file loaded to the system as mentioned before. Thus, in order to successfully load a new pipeline, the corresponding XML file should contain all the required values. These values include:

- KML file path, containing the geographical profile of the pipeline element, such as valves and stations,
- Optimization formulation file path, containing optimization parameters, such as the pipeline hydraulic characteristics and other operational constraints,
- SCADA data files path, if such data exists,
- Pump coefficients for calculating speed, if the pump is variable speed.



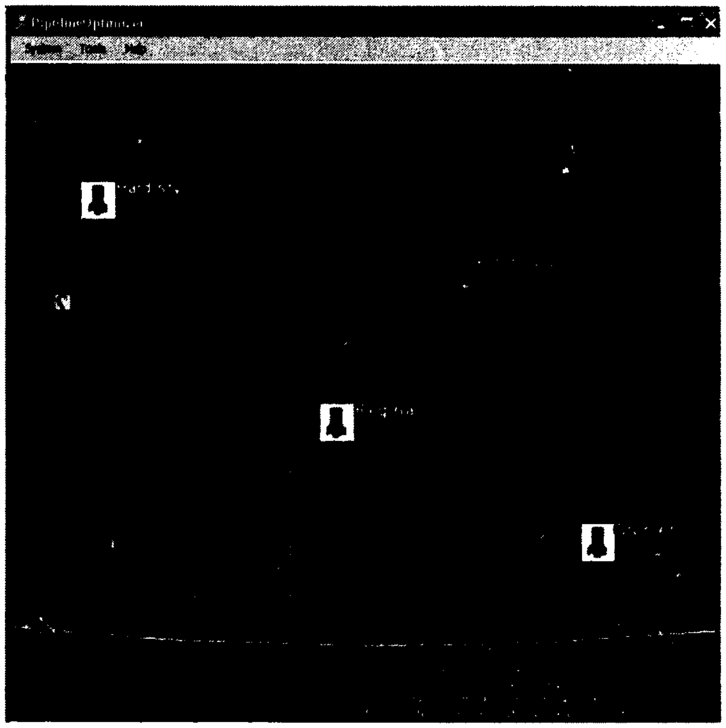


Figure 53: Pembina pipeline (top), hypothetical pipeline (bottom)

8.4 Commercialization of the system

Optimizing pipeline operation is considered one of the important problems of the pipeline operators, as it can significantly impact the amount of their profit. Similar to our industrial partner, other pipeline operator companies can potentially be interested in utilising a software system, which can help them optimize and improve their operational routines. As the application is developed with the primary goal of providing the pipeline operators with the optimized pipeline configuration, we were interested in commercializing the software to have the opportunity of collaborating with more industrial partners. This collaboration can also aid us to make the software a better fit for the needs of the industry, by being presented to particular demands of different companies.

This also can introduce new challenges in the deployment of engineering software for the oil industry, which can be published in academia to be available for other researchers in similar context.

With having these in mind, in May 2011 we started to commercialize the application with the help of a business partner. The business partner is responsible to identify potential industrial partners and build the ground for presenting the application. The demo of the application was built and recently presented to several companies in Calgary, who showed initial interest in collaboration. The follow up meetings with these companies is being planned and after finalizing the details of the collaboration, the software will be customized accordingly to meet their needs.

8.5 **Chapter Summary**

In this chapter, the usage scenarios of the system were presented to provide some insight on how the developed software can assist the operators and managers in making the right decisions. The impact of pipeline parameters changes, such as delivery volume and power rates, on the total operation cost were discussed. Besides the mentioned parameters, other parameters of the pipeline formulation file can be altered as required and the resulting impact can be visualised and studied using the charting utility integrated to the system.

Speed charts, providing operational pump speed values to the user for operating the pipeline with optimized configuration were presented. Loading a new pipeline, which is another feature of the developed system, were also introduced in this chapter.

Next Chapter provides the summary, conclusions and future directions of this thesis.

Chapter Nine: **Summary, Conclusions and Future Works**

This chapter summarizes and concludes the thesis. The summary of the thesis is presented in Section 9.1, followed by the concluding remarks in Section 9.2 and suggestions on the potential research trends in the future to follow up this work, presented in Section 9.3.

9.1 **Summary**

This thesis included two major components. (1) The SLR on software engineering for SES software development and, (2) the experiences in building an engineering software in which the ideas inspired from the SLR were applied. The SLR aimed at assessing the state of the art and practice in software engineering for SES development and in particular identifies weaknesses and strengths, highlights challenges and finds potential future research trends from the perspective of developers, researchers and scientists. The case study aimed at developing an industrial pipeline operation optimization software and decision support system.

In order to provide the ground for conducting this study, in Chapter 2, the background on software engineering practices and previous publications in the field were presented. Also, a group of common software systems from the oil industry family were introduced.

In Chapter 3, we presented the steps taken to conduct the SLR and systematic mapping on software engineering for SES development. SLRs were mentioned in the literature to be one of the very common types of EBSE providing valuable information for scientists and practitioners, which are methodical, comprehensive and organized reviews about the state of the art in a particular domain and about a certain subject. Systematic mappings were also identified as a proper starting point for more detailed studies as they categorize different types of primary studies and give summary of the results.

A standard methodology for conducting systematic literature reviews was employed using the well-known digital libraries in the field and the relevant information from the publications based on a group of research questions were extracted and integrated, forming the main part of the review protocol. The best practices reported in literature were identified as applicable to various problem domains.

In Chapter 4, in the process of developing our industrial engineering software for the optimization of oil pipeline operation, as a part a major ongoing research project, we brought the insights taken from systematic literature review into practice. In the case study we conducted, as a second goal, we aimed at testifying the challenges of SES development and to verify the applicability of the best practices, investigate their adaptability and validate the solutions reported, where relevant. This industrial collaboration built the ground to enhance the quality and reliability of the end product while optimized the cost and time issues by providing us with the highlights on the potential bottlenecks and available solutions upfront.

The main goal of the case study was the development of the engineering software and decision support system to provide the optimal operation scheduling for the operators as well as the possibility to visually inspect the pipeline important variables such as total power consumption, power cost for each of the stations and the pump speeds in variable speed pump stations. In the beginning of the project, the domain terminology were identified to efficiently communicate with the domain experts and to understand the pipeline system basics. This terminology includes the definition of pipeline elements such as valves, pumps, and pump stations which are the main components of each pipeline system. These elements were represented in the optimization formulation script by certain variables (devised by another student in SoftQual research group). The optimization objective, which is the minimization of the pipeline operation cost, was solved by the commercial solver embedded in the application. Also as mentioned before as a second goal by building this engineering software, we aimed at applying the ideas inspired by the SLR.

In Chapter 5, the oil pipeline operation application requirements were presented as well as the analysis and design documents used in designing the system. We adopted object-oriented methodology for developing this system. System actors, use-case diagram and activity diagrams such as calling the optimization solver, viewing charts and loading a new pipeline were presented. MVC architectural pattern was adopted in order to support testability and maintainability besides better managing different levels of the application. Different elements of the system based on MVC architecture, which include the system entities, controllers such as optimization controller and application views which are different windows of the system were tabulated and described.

In Chapter 6, the iterative development process used to develop the system was introduced. Dependency analysis, in order to extract the dependencies among the system artefacts, was performed. This analysis identifies the highly dependent elements, whish later gives the developer a precise idea about the element relationships, and is helpful when any artefact of the application is required to be re-used, upgraded or replaced. Some of the application metrics such as lines of code, IL instructions and cyclomatic complexity was calculated and reported followed by presenting selected dependency graphs.

In Chapter 7, the testing practices undertook in order to detect system defects and assuring the system correct functionality was discussed. Black box unit testing was employed. The source code symbol and branch coverage values were also reported. In order for testing the functionality of the system through its GUI, record and play back GUI testing method was used. As the last section in that chapter, mutation testing was applied on the optimization formulation script, in order to fortify the sensitivity analysis done by another member of SoftQual research group.

In Chapter 8, the usage scenarios of the system were presented to provide insight on how the developed software can assist the operators and managers in making the right decisions. The impact of pipeline parameters changes, such as delivery volume and power rates, on the total operation cost were discussed. Speed charts, providing operational pump speed values to the pipeline operator for operating the pipeline with optimized configuration were presented. Loading a new pipeline, which is another feature of the developed system, was also introduced in that chapter.

9.2 Conclusions

Software systems are one of (if not) the most critical parts of any modern system (e.g., scientific, engineering, health-care, and military). Traditionally, scientists and engineers have used ad-hoc programming and software development techniques (e.g., code and debug) to develop their required software systems. However, with advances in different areas of software engineering, more and more software engineering concepts, tools and methodologies are being adopted and used by scientists and engineers in their software development tasks.

We aimed at providing a comprehensive background on identifying major issues in software engineering for SES. By conducting a systematic literature review, we aimed to systematically extracting available insights and inspirations from the literature and to come up with a structured guideline on how to improve the whole development process of SES.

By systematically reviewing and categorizing 83 selected papers in the area published from 1980 to 2010, we were able to extract and report interesting information about how SES is being developed in various domains. The demographic data presented provide interesting insights about the research, researchers and domains of SES projects.

We found the trend of the publications to be increasing between 1980 and 2010. This shows that this area in recent years is gaining more attentions. By extracting the breakdown of research affiliates in this field, we found that not surprisingly the university is the lead in publishing on SES development followed by research groups which conduct collaborative works among the university and private sector. Other publications are coming from research groups affiliated with the government. All of these groups are increasing their publications in the recent years.

Physics and biology are two domains from which we had most of the publications, as these two disciplines require software for dealing with their complex simulations and modeling.

Most of the publications were focused on design, architecture, testing and context-dependent methodologies and solutions to deal with the complexities and challenges of developing SES.

By conducting the SLR we could characterize SES as a type of software which has four main differences from commercial software. First, finalizing the requirements in SES development is not practical and possible, as in most cases the goal of developing the software is to find the solution to a problem for which no prior solution exists. Thus, the requirement elicitation will remain an ongoing process throughout the life cycle of the software. Actually developing working software quickly, is reported in the literature to be a treatment for extracting the requirements more precisely at the later stages of the development.

Second, as the main objective of developing SES is to provide a correct and reliable code which can be utilized to improve the target science or engineering discipline. As a result, the factor of building a working system in the shortest amount of time outweighs adopting rigorous software engineering practices to ensure the quality of end product. This is shown in the literature to be the main reason of ignoring most of the beneficial practices in software engineering, such as considering testability, reusability and maintainability in the design of SES.

Third, the developers of SES are mostly domain experts (i.e. scientists and engineers). Not surprisingly, they are not academically trained to develop software similar to software engineers. This makes adopting software engineering practices more challenging for the domain experts.

Finally, testing SES has two independent stages of verification and validation. First stage is testing the scientific/engineering core for which usually no certain test oracle exists. This in particular is a very context-dependent and challenging practice. Second stage is testing the software that provides access to that scientific/engineering core, and can be performed following the common testing practices.

The mentioned characteristics often introduce certain challenges in the development of SES, which we aggregated besides the solutions provided, if any. These challenges reported in the thesis also suggest the area of improvement and further research for future projects in different context. The extent of the match between reality of SES development and what is expected to be followed as suggested by the software engineering methodologies show the current state of the art and gives a glimpse of what is happening in practice.

We were able to identify best practices to give practical insights to the developers who want to take the advantages of previous experiences and to adopt applicable guidelines suggested by other developers in their own practice. Strengths and weaknesses reported in the publications may pave the way for conducting a careful and precise development practice. Upcoming trends in SES development propose the areas of investments for further research and practice.

By developing industrial engineering software, we experienced the particular challenges of this field and tried the practicality of the proposed solutions to improve the development process besides enhancing the quality of the end product. We concluded that developing SES is different from conventional software development in practices mostly because its primary aim is to help the scientists and engineers better understand, analyze and resolve their domain issues and thus is highly tied with the knowledge and expertise of scientists as the real owners of the software.

Our case study confirmed the observations reported by other practitioners in different stages of the development, such as the difficulty of requirement elicitation and the complexity of the testing. Although we aimed at developing software for pipeline operation optimization, we had to deal with two segments in the system: the engineering core responsible for finding the optimal operation settings and the software which was developed to utilize the engineering core and provide the decision support facilities. We could follow software engineering practices to facilitate the challenges of the interface software, as shown in Table 29. However, certain challenges of designing, implementing and testing the engineering core could not be resolved by following common software engineering practices.

Challenge	**Solutions adopted for developing the engineering core**	**Solutions adopted for developing the software interface**
Requirement elicitation	Frequent meetings with domain expert, to learn the domain expertise and verify understanding of the optimization problem	-Meetings with the domain experts to understand how the software will be utilized -Adopting iterative development approach
Design		OO technology and MVC architectural pattern
Implementation	Using Lingo for implementing MILP technique	Using Visual studio 2008 integration environment and C# language for coding
Testing	- Sensitivity analysis - Solving the optimization problem using genetic algorithm	- Unit testing - GUI testing
Maintenance		Dependency analysis

Table 29: Summarizing the solutions adopted for developing the engineering core and solutions adopted for the software interface

It is worth mentioning that the solutions summarized in the second column of the table is reported based on the experience of another member of our research group, who was in charge of developing the optimization module (i.e. engineering core).

To conclude we need to emphasize that without having proper understanding of the domain and without tightly interacting and communicating with domain experts and scientists, the scientific software development would not be a reliable and precise practice. A lot of practical work has been done and evaluated by experts for improving the practice of developing SES to ensure the reliability and robustness of the end product, yet there exists certain challenges which need to be addressed.

9.3 **Future Works**

Using the results we have gained from the systematic literature review, we plan to conduct surveys with practitioner SES developers to solicit the latest trends, challenges and needs in those communities to and to identify more practical and empirical evidence from other ongoing projects.

While searching to collect our primary studies, we found several insightful books on the topic, that we did not include in this study. We plan to extend the scope of our study by considering the related books at a later stage of this research.

Considering the reasonable trade-off between the required time and effort on one side and the completeness of the results on the other side, we did not include in our search keywords the domain names of scientific and engineering disciplines. Thus some of the domain-specific publications may exist, which are not currently included in the SLR. We plan to expand our search keywords to find and include such papers in the SLR at a later stage.

In the software application developed, we plan to integrate a database management system, instead of using text files, in order to manage the historical data taken from industry and the data generated by the optimization engine in a much more efficient manner.

In order to improve the efficiency and to minimize the operation cost of oil distribution systems, other features, such as batch-scheduling optimization and reduction of pump maintenance costs can be added to the system based on the demand taken from other potential industrial partners. Also in the optimization module, software engineering aspects such as maintainability and readability can be improved.

As the system was developed as a sample of a SES in this study, the main focus has been given to the proper design and correct functionality of the system by following standard software engineering practices, thus the usability and user-friendliness of the system through its GUI can be further enhanced and customized based on the ideas taken from particular companies which will utilize the software in real practices.

References

[1] "Software Horror Stories," in *http://www.cs.tau.ac.il/~nachumd/horror.html*, Last accessed: July 2010.

[2] "Toyota's lesson: Software can be unsafe at any speed," in *http://blogs.computerworld.com/15547/toyotas_lesson_software_can_be_unsafe_at_any_speed*, Last accessed: July 2010.

[3] D. F. Kelly, "A Software Chasm: Software Engineering and Scientific Computing," IEEE Software, vol. 24, no. 6, pp. 118-119, 2007.

[4] G. Wilson, "Those who will not learn from history," Computing in Science and Engineering, vol. 10, no. 3, pp. 5-6, 2008.

[5] G. V. Wilson, "Where's the real bottleneck in scientific computing?," American Scientist, vol. 94, no. 1, pp. 5, 2006.

[6] J. Segal, "Models of scientific software development," in *Workshop on Software Engineering in Computational Science and Engineering*, 2008.

[7] J. F. Cremer, R. S. Palmer, and R. E. Zippel, "Creating scientific software," Transactions of the Society for Computer Simulation International, vol. 14, no. 1, pp. 37 - 49, 1997.

[8] J. Segal, "Scientists and software engineers: A tale of two cultures," in *Psychology of Programming Interest Group*, 2008.

[9] B. Boehm, "Managing Software Productivity and Reuse," in *Computer Physics Communications*. vol. 32, 1999.

[10] S. M. Easterbrook and T. C. Johns, "Engineering the Software for Understanding Climate Change," Computing in Science and Engineering, vol. 11, no. 6, pp. 65-74, 2009.

[11] "Reservoir Engineering Software and Services," in *http://www.fekete.com*, Last accessed: July 2010.

[12] "Energy Solution International," in *http://www.energy-solutions.com/*, Last accessed: July 2010.

[13] "Engineering Software Center," in *http://www.engsoftwarecenter.com/*, Last accessed: July 2010.

[14] "Intuitive Software for Structural Engineering," in *http://www.iesweb.com/*, Last accessed: July 2010.

[15] "Workshop on Software Engineering in Health Care," in *http://www-swe.informatik.uni-heidelberg.de/sehc09/index.htm*, Last accessed: July 2010.

[16] "International Workshop on Aerospace Software Engineering," in *http://crisys.cs.umn.edu/icse-workshop/Program.htm*, Last accessed: July 2010.

[17] "International Workshop on Software Engineering for Computational Science and Engineering," in *http://www.cs.ua.edu/~SECSE10*, Last accessed: July 2010.

[18] "Workshop on Software Research and Climate Change " in *http://www.cs.toronto.edu/wsrcc/*, Last accessed: July 2010.

[19] "Workshop on Software Engineering for Automotive Systems " in *http://www.inf.ethz.ch/personal/pretscha/events/seas07/*, Last accessed: July 2010.

[20] J. segal, "Some Problems of Professional End User Developers," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2007.
[21] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software engineering," in *Evidence-Based Software Engineering*, 2007.
[22] B. Kitchenham, P. Breretona, D. Budgenb, M. Turnera, J. Baileyb, and S. Linkmana, "Systematic literature reviews in software engineering–A systematic literature," Information and Software Technology, vol. 51, no. 1, pp. 7-15, 2009.
[23] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," in *International conference on Software Engineering*, 2007, pp. 550-559.
[24] L. Hochstein and V. R. Basili, "The ASC-Alliance Projects: A Case Study of Large-Scale Parallel Scientific Code Development," Computer, vol. 41, no. 3, pp. 50-58, 2008.
[25] S. Smith, "Systematic Development of Requirements Documentation for General Purpose Scientific Computing Software," in *IEEE International Requirement Engineering Conference*, 2006.
[26] J. Tang, "Developing scientific computing software, current processes and future directions," Master thesis, McMaster university, 2009.
[27] "The R Project for Statistical Computing," in http://www.r-project.org/, Last accessed: April 2011.
[28] "Gretl," in http://gretl.sourceforge.net/, Last accessed: July 2010.
[29] R. Kendall, J. C. Carver, D. Fisher, D. H. A. Mark, D. Post, C. E. R. Jr, and S. Squires, "Development of a Weather Forecasting Code: A Case Study," IEEE Software, vol. 25, no. 4, pp. 59-65, 2008.
[30] S. Kumar, K. Tamura, and M. Nei, "MEGA3: Integrated software for Molecular Evolutionary Genetics Analysis and sequence alignment," Briefings in Bioinformatics vol. 5, no. 2, pp. 150-163, 2004.
[31] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics," Genome biology, vol. 5, no. 10, pp. R80, 2004.
[32] J. C. Nelson, "QGENE: software for marker-based genomic analysis and breeding," Molecular Breeding, vol. 3, no. 1, pp. 239-245, 1997.
[33] J. Segal and C. Morris, "Developing Scientific Software," IEEE Software, vol. 25, no. 4, pp. 18-20, 2008.
[34] Basili, V. R. Carver, J. C. Cruzes, D. Hochstein, L. M. Hollingsworth, J. K. Shull, F. Zelkowitz, and M.V, "Understanding the High-Performance-Computing Community:A Software Engineer's Perspective," IEEE Software, vol. 25, no. 4, pp. 29-36, 2008.
[35] P. K. Chilana, C. L. P, and A. J. Ko, "Comparing Bioinformatics Software Development by Computer Scientists and Biologists: An Exploratory Study," in

Workshop on Software Engineering for Computational Science and Engineering, 2009.

[36] V. Maxville, "Preparing Scientists for Scalable Software Development," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009.

[37] Okoli C. and S. K., "A Guide to Conducting a Systematic Literature Review of Information Systems Research," Working Papers on Information Systems, vol. 10, no. 26, pp. 2010.

[38] A. Fink, *Conducting research literature reviews: from the Internet to paper*: Sage Publications Ltd., 2005.

[39] B. Kitchenham, P. Brereton, and D. Budgen, "The Educational Value of Mapping Studies of Software Engineering Literature," in *ACM/IEEE Conference on Software Engineering*, 2010.

[40] Barbara A. Kitchenham, Tore Dyba, and Magne Jorgensen, "Evidence-Based Software Engineering," in *International Conference on Software Engineering*, 2004.

[41] D. S. Cruzes and T. Dybå, "Synthesizing evidence in software engineering research," in *ACM-IEEE Symposium on Empirical Software Engineering and Measurement*, 2010.

[42] S. E. Campbell, D. G. Seymour, and W. R. Primrose, "A systematic literature review of factors affecting outcome in older medical patients admitted to hospital," Age and Ageing, vol. 33, no. 1, pp. 110-115, 2004.

[43] Pittaway, Luke, Robertson, Maxine, Munir, Kamal, Denyer, David, Neely, and A. D, "Networking and Innovation: A Systematic Review of the Evidence," International Journal of Management Reviews, vol. 5, no. 3, pp. 137-168, 2004.

[44] T. Baines, H. Lightfoot, G. M. Williams, and R. Greenough, "State of the art in lean design engineering: a literature review on white collar lean," Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, vol. 220, no. 9, pp. 1539-1547, 2006.

[45] S. Ali, L. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-based Test-Case Generation," in *IEEE Transactions on Software Engineering*, 2009.

[46] M. Harman, S. A. Mansouri, and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications," in *Department of Computer Science King's College London, Technical Report TR-09-03*, 2009.

[47] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," Information and Software Technology, vol. 52, no. 1, pp. 14-30, 2010.

[48] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software engineering," in *Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71-80.

[49] D. Feitosa, K. R. Felizardo, L. B. R. d. Oliveira, D. Wolf, and E. Y. Nakagawa, "Software Engineering in the Embedded Software and Mobile Robot Software

Development: A Systematic Mapping," in *International Conference on Software Engineering and Knowledge Engineering*, 2010.

[50] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How Do Scientists Develop and Use Scientific Software?," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 1-8.

[51] C. Greenough and D. J. Worth, "Computational science and engineering department software development best practice," in *Technical report ral-tr-2008-022, SFTC Rutherford AppletonLaboratory*, 2008

[52] "SCHEDULE++ PIPELINE&PORT," in *http://www.oilindustryscheduling.com/pipeline_port.html*, Last accessed: April 2011

[53] "Planning, Scheduling and Blending Optimization Software," in *http://www.m3tch.com/simto-products.html*, Last accessed: April 2011.

[54] "PEDI," in *http://www.ssischeduling.com/*, Last accessed: April 2011.

[55] "H/SCHED," in *http://www.haverly.com/OmniSuite.htm*, Last accessed: April 2011.

[56] "High Performance SAP-Primavera Integration - Pipeline," in *www.pipelinesoftware.com/psi/pipeline_p3e.jsp*, last accessed: April 2011

[57] V. Garousi and T. Varma, "A Bibliometric Assessment of Canadian Software Engineering Scholars and Institutions (1996-2006)," Canadian Journal on Computer and Information Science, vol. 3, no. 2, pp. 19-29, 2010.

[58] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," in *IEEE Transactions on Software Engineering*, 2010.

[59] R. L. Glass, "An assessment of systems and software engineering scholars and institutions," Journal of Systems and Software, vol. 27, no. 1, pp. 63-67, 1994.

[60] G. Wilson, "Software carpentry " Computing in science and engineering, vol. 8, no. 6, pp. 66-69, 2006.

[61] b. M. B. Blake, "A Student-Enacted Simulation Approach to Software Engineering Education," IEEE Transactions on Education, vol. 46, no. pp. 124-132, 2001.

[62] O. Bilhan, M. E. Emiroglu, and O. Kisi, "Application of two different neural network techniques to lateral outflow over rectangular side weirs located on a straight channel," Advances in Engineering Software, vol. 41, no. 6, pp. 831-837, 2010.

[63] K. L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," IEEE Transactions on Software Engineering, , vol. 6, no. 1, pp. 2-13, 1980.

[64] J. Segal, A. Grinyer, and H. Sharp, "The type of evidence produced by empirical software engineers," in *Workshop on Realising Evidence-based Software Engineering*, 2005.

[65] M. Rodgers, A. Sowden, M. Petticrew, L. Arai, H. Roberts, N. Britten, and J. Popay, "Testing Methodological Guidance on the Conduct of Narrative Synthesis in Systematic Reviews," Evaluation, vol. 15, no. 1, pp. 49-74, 2009.

[66] M. E. Larsson and P. A. Laplante, "On the Complexity of Design in Imaging Software," in *IEEE International Conference on Engineering of Complex Computer Systems*, 2006, pp. 37 - 46.

[67] A. Spinelli, P. Salvaneschi, M. Cadei, and M. Rocca, "MI—an object oriented environment for integration of scientific applications," in *Conference on Object-oriented programming systems, language and applications*, 1994, pp. 212 - 222.

[68] S. Smith and W. Yu, "A document driven methodology for developing a high quality Parallel Mesh Generation Toolbox," Advances in Engineering Software, vol. 40, no. 11, pp. 1155-1167, 2009.

[69] S. F. Siegel and L. F. Rossi, "Analyzing BlobFlow: A Case Study Using Model Checking to Verify Parallel Scientific Software," Recent advances in parallel virtual machine and message passing interface, vol. 5205, no. 1, pp. 274-282, 2008.

[70] D. Kelly and R. Sanders, "Assessing the quality of scientific software," in *Workshop on Software Engineering for Computational Science & Engineering*, 2008.

[71] J. R. Cary, S. G. Shasharina, J. C. Cummings, J. V. W. Reynders, and P. J. Hinker, "Comparison of C++ and Fortran 90 for object-oriented scientific programming," Computer Physics Communications, vol. 105, no. 1, pp. 20-36, 1997.

[72] T. L. Veldhuizen and M. E. Jernigan, "Will C++ be faster than Fortran?," in *Scientific Computing in Object-Oriented Parallel Environments*, 1997, pp. 49-56.

[73] T. Veldhuizen, "Scientific Computing: C++ Versus Fortran," in http://www.drdobbs.com/184410315, Last accessed: Aprill 2011.

[74] D. Kelly, D. Hook, and R. Sanders, "Five Recommended Practices for Computational Scientists Who Write Software," Computing in Science and Engineering, vol. 11, no. 5, pp. 48-53, 2009.

[75] D. Hook and D. Kelly, "Testing for trustworthiness in scientific software," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 59-64.

[76] D. Hook and D. Kelly, "Mutation Sensitivity Testing," IEEE Design & Test, vol. 11, no. 6, pp. 40-47, 2009.

[77] R. Sanders and D. Kelly, "The Challenge of Testing Scientific Software," in *Conference for the Association for Software Testing*, 2008.

[78] R. Sanders and D. Kelly, "Dealing with Risk in Scientific Software Development," IEEE Software, vol. 25, no. 4, pp. 21-28, 2008.

[79] J. Segal, "When Software Engineers Met Research Scientists: A Case Study," Empirical Software Engineering, vol. 10, no. 1, pp. 517-536, 2005.

[80] J. Segal, "Some challenges facing software engineers developing software for scientists," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009.

[81] J. Segal, "Software Development Cultures and Cooperation Problems: A Field Study of the Early Stages of Development of Software for a Scientific Community," Computer Supported Cooperative Work, vol. 18, no. 5-6, pp. 581–606, 2009.

[82] R. Kendall, D. Post, and J. I. c. S. D. L. I. C. S. Andrew Mark "Case Study of the NENE Code Project," IEEE Design & Test vol. 12, no. 3, pp. 28-33, 2010.

[83] S. L. Eddins, "Automated Software Testing for Matlab," Computing in Science and Engineering, vol. 11, no. 6, pp. 48-55, 2009.

[84] D. Hook, "Using code mutation to study code faults in scientific software," Master's thesis, Queen's University, 2009.

[85] K. Frounchi, L. C. Briand, L. Grady, Y. Labiche, and R. Subramanyan, "Automating Image Segmentation Verification and Validation by Learning Test Oracles," in *Carleton University, Technical Report SCE-09-06*, 2009.

[86] C. A. Crabtree, A. G. Koru, C. Seaman, and H. Erdogmus, "An Empirical Characterization of Scientific Software Development Projects According to the Boehm and Turner Model: a Progress Report," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009.

[87] R. A. Bartlett, "Integration strategies for Computational Science & Engineering software," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 35-42.

[88] M. A. Heroux and J. M. Willenbring, "Barely Sufficient Software Engineering:10 Practices to Improve Your CSE Software," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009.

[89] C. Macaulay, D. Sloan, X. Jiang, P. Forbes, S. Loynton, J. R. Swedlow, and P. Gregor, "Usability and User-Centered Design in Scientific Software Development," IEEE Software, vol. 26, no. 1, pp. 96-102, 2009.

[90] D. D. Roure and C. Goble, "Software design for empowering scientists," IEEE Computer Society, vol. 26, no. 1, pp. 88 - 95, 2009.

[91] G. Fischer, K. Nakakoji, and Y. Ye, "Metadesign: Guidelines for Supporting Domain Experts in Software Development," IEEE Software, vol. 26, no. 5, pp. 37-44, 2009.

[92] D. Woollard, C. Mattmann, and N. Medvidovic, "Injecting Software Architectural Constraints into Legacy Scientific Applications," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 65-71.

[93] R. Arora, P. Bangalore, and M. Mernik, "Developing Scientific Applications Using Generative Programming," in *Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 51-58.

[94] D. Woollard, N. Medvidovic, Y. Gil, and C. A. Mattmann, "Scientific Software as Workflows: From Discovery to Distribution," IEEE Software, vol. 25, no. 4, pp. 37-43, 2008.

[95] K. S. Ackroyd, S. H. Kinder, G. R. Mant, M. C. Miller, C. A. Ramsdale, and P. C. Stephenson, "Scientific Software Development at a Research Facility," IEEE Software, vol. 25, no. 4, pp. 44-51, 2008.

[96] M. Vigder, N. G. Vinson, J. Singer, D. Stewart, and K. Mews, "Supporting Scientists' Everyday Work: Automating Scientific Workflows," IEEE Software, vol. 25, no. 4, pp. 52-58, 2008.

[97] S. A. Vilkomir, W. T. Swain, J. H. Poore, and K. T. Clarno, "Modeling Input Space for Testing Scientific Computational Software: A Case Study," in *International conference on Computational Science*, 2008, pp. 291 - 300.

[98] J. P. Kenny, C. L. Janssen, M. S. Gordon, M. Sosonkina, and T. L. Windus, "A component approach to collaborative scientific software development: Tools and techniques utilized by the Quantum Chemistry Science Application Partnership," Scientific Programming, vol. 16, no. 4, pp. 287-296, 2008.

[99] B. A. Allan, B. Norris, W. R. Elwasif, and R. C. Armstrong, "Managing scientific software complexity with Bocca and CCA," Scientific Programming vol. 16, no. 4, pp. 315-327, 2008.

[100] J. C. Carver, "Post-Workshop Report for the Third International Workshop on Software Engineering for High Performance Computing Applications," Sigsoft Software Eng. Notes, vol. 32, no. 5, pp. 38-43, 2007.

[101] R. P. Kendall, D. E. Post, J. C. Carver, D. B. Henderson, and D. A. Fisher, "A Proposed Taxonomy for Software Development Risks for High-Performance Computing (HPC) Scientific/Engineering Applications," Technical report CMU/SEI-2006-TN-039, Carnegie Mellon 2007.

[102] S. Smith, L. Lai, and R. Khedri, "Requirements Analysis for Engineering Computation: A Systematic Approach for Improving Reliability," Reliable Computing vol. 13, no. 1, pp. 83-107, 2007.

[103] S. Baxter, S. W. Day, J. S. Fetrow, and S. J. Reisinger, "Scientific Software Development Is Not an Oxymoron," PLoS Computational Biology, vol. 2, no. 9, pp. 87, 2006.

[104] D. W. Kane, M. M. Hohman, E. G. Cerami, M. W. Mccormick, K. F. Kuhlmman, and J. A. Byrd, "Agile methods in biomedical software development: a multi-site experience report," in *Bioinformatics*, 2006.

[105] M. Broy, "Challenges in Automotive Software Engineering," in *International conference on Software engineering*, 2006, pp. 33 - 42.

[106] C. E. Rasmussen, M. J. Sottile, S. S. Shende, and A. D. Malony, "Bridging the language gap in scientific computing: the Chasm approach," Concurrency and computation: practice and experience, vol. 18, no. 1, pp. 151-162, 2006.

[107] D. Kane, "Introducing Agile Development into Bioinformatics: An Experience Report," in *Proceedings of the Conference on Agile Development*, 2003, pp. 132-140.

[108] W. A. Wood and W. L. Kleb, "Exploring XP for Scientific Research," IEEE Software, vol. 20, no. 3, pp. 30-36, 2003.

[109] E. Houstis, E. Gallopoulos, R. Bramley, and J. Rice, "Problem-Solving Environments for Computational Science," IEEE Computational Science & Engineering, vol. 4, no. 3, pp. 18 - 21, 1997.

[110] A. Dall'Osso, "Using computer algebra systems in the development of scientific computer codes," Future Generation Computer Systems, vol. 19, no. 2, pp. 143-160, 2003.

[111] X. Jiao, M. T. Campbell, and M. T. Heath, "Roccom: an object-oriented, data-centric software integration framework for multiphysics simulations," in *International conference on Supercomputing*, 2003, pp. 358 - 368.

[112] P. M. Johnson, "Second international workshop on software engineering for high performance computing system applications," in *Conference on Software engineering*, 2005, p. 683.

[113] P. Johnson, "Workshop on software engineering for high performance computing system (HPCS) applications," in *Conference on Software Engineering*, 2004, p. 772.
[114] F. M. Hovenden, S. D. Walker, H. C. Sharp, and M. Woodman, "Building quality into scientific software," Software Quality Journal, vol. 5, no. 1, pp. 25-32, 1996.
[115] S. Smith and L. Lai, "A New Requirements Template for Scientific Computing," in *Workshop of Situational Requirements Engineering Processes*, 2005.
[116] C. Blilie, "Patterns in Scientific Software: An Introduction," Computing in Science and Engineering, vol. 4, no. 3, pp. 48-53, 2002.
[117] T. v. d. Wal, Knapen, Svensson, Athanasiadis, and Rizzoli, "Trade-offs in the design of cross-disciplinary software systems," in *International congress on modeling and simulation; advances and applications for management and decision making*, 2005.
[118] H. Gardner, "Design Patterns in Scientific Software," in *Computational Science and Its Applications*, 2004, pp. 776–785.
[119] T. Cickovski, T. Matthey, and J. u. A. Izaguirre, "Design Patterns for Generic Object-Oriented Scientific Software," in *International Conference on Software Engineering*, 2005.
[120] C. Letondal and U. Zdun, "Anticipating Scientific Software Evolution as a Combined Technological and Design Approach," in *International Workshop on Unanticipated Software Evolution*, 2003
[121] A. Gupta, N. Dubey, D. Naidu, P. Neethinathan, T. P. Srinivasan, B. G. Krishna, R. Nandakumar, and P. K. Srivastava, "Designing Satellite Data Processing Software Systems Using Object Oriented Technology," in *Indian Cartographer*, 2002, pp. 49-54.
[122] S. Z. Guyer and C. Lin, "Broadway: A Software Architecture for Scientific Computing," in *The Architecture of Scientific Software*, 2000, pp. 175-192.
[123] D. C. Arnold and J. J. Dongarra, "Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications," in *working conference on the architecture of scientific software*, 2000, pp. 39-55.
[124] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High-Performance Scientific Computing," in *IEEE International Symposium on High Performance Distributed Computing*, 1999, pp. 13-24.
[125] D. E. Bernholdt, W. R. Elwasif, and J. A. Kohl, "Communication Infrastructure in High-Performance Component-Based Scientific Computing," in *European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2002, pp. 260 - 270.
[126] T. Epperly, S. R. Kohn, and G. Kumfert, "Component technology for high-performance sceintific simulation software," in *Working Conference on the Architecture of Scientific Software*, 2000, pp. 69 - 86.
[127] D. E. Bernholdt, R. C. Armstrong, and B. A. Allan, "Managing complexity in modern high end scientific computing through component-based software engineering," in *Workshop on Productivity and Performance in High-End Computing*, 2004.

[128] L. Hatton, "The T-Experiments: Errors in scientific software," IEEE Computational Science & Engineering, vol. 4, no. 2, pp. 27-38, 1997.
[129] D. E. Post and R. P. Kendall, "Software Project Management and Quality Engineering Practices for Complex, Coupled Multiphysics, Massively Parallel Computational Simulations: Lessons Learned From ASCI," International Journal of High Performance Computing Applications, vol. 18, no. 4, pp. 399 - 416, 2004.
[130] D. M. Beazley, "Automated scientific software scripting with SWIG," Future Generation Computer Systems, vol. 19, no. 5, pp. 599-609, 2003.
[131] D. M. Beazley and P. S. Lomdahl, "Feeding a Large-scale Physics Application to Python," in *International Python Conference*, 1997 pp. 21-28.
[132] "Software Engineering of Energy-related Systems," in *http://www.ucalgary.ca/~vgarousi/project-sw-energy.html*, Last accessed: July 2010.
[133] D. Stutz, T. Neward, and G. Shilling, *Shared Source CLI Essentials*: O'Reilly, 2003.
[134] E. Abbasi and V. Garousi, "An MILP-based Formulation for Minimizing Pumping Energy Costs of Oil Pipelines: Beneficial to both the Environment and Pipeline Companies," in *press, Springer Journal on Energy Systems*, 2010.
[135] L. systems, "LINGO 12.0 " in *Manual for Optimization Modeling Software for Linear, Nonlinear, and Integer Programming*, 2009.
[136] D. E. Perry, "Lecture 11: Validity," in *http://users.ece.utexas.edu/~perry/education/382c/handouts/L11.pdf*, Last accessed: April 2011.
[137] "Pembina Pipeline Corporation," in *http://www.pembina.com*, Last accessed: April 2011.
[138] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical Software Engineering, vol. 14, no. 2, pp. 131-164, 2009.
[139] E. Abbasi, "Development and industrial application of an MILP-based optimisation algorithm fr minimizing pumping cost and carbon footprint of oil pipelines," University of Calgary, Master's thesis, 2010.
[140] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," vol. 25, no. 4, pp. 557 - 572, 1999.
[141] "Alaska pipeline," in *http://www.solarstorms.org/Spipeline.html*, Last accessed: April 2011.
[142] "Enbridge Pipeline System," in *http://en.wikipedia.org/wiki/Enbridge_Pipeline_System*, Last accessed: April 2011.
[143] "Pump curve," in *http://www.unicade.com/cmax/BuyPump.htm*, Last accessed: April 2011.
[144] "Pump station," in *http://static.guim.co.uk/sys-images/Guardian/Pix/pictures/2007/04/12/Gazprom372.jpg*, Last accessed: April 2011.

[145] "Oil pipeline control valves " in *http://my.qoop.com/store/Photogenic-Asia---Royalty-Free-Images-320125827136010 0/Oil-Pipeline-Control-Valves-by-Shi-Yali-qpps_559727812410191/*, Last accessed: april 2011.
[146] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*: Wiley-Interscience, 1999.
[147] "LINDO Systems - Optimization Software: Integer Programming, Linear Programming, Nonlinear Programming, Stochastic Programming, Global Optimization," in *http://www.lindo.com/*, Last accessed: April 2011.
[148] "KML," in *http://en.wikipedia.org/wiki/Keyhole_Markup_Language*, Last accessed: April 2011.
[149] "COM," in *http://www.microsoft.com/com/default.mspx*, Last accessed: April 2011.
[150] "MVC vs. MVP," in *http://dotnetslackers.com/articles/designpatterns/Evergreen-but-still-topical-MVC-vs-MVP.aspx*, Last accessed:April 2011.
[151] "Class diagrams," in *http://en.wikipedia.org/wiki/Class_diagram*, Last accessed: April 2011.
[152] "Iterative and incremental development," in *http://en.wikipedia.org/wiki/Iterative_and_incremental_development*, Last accessed: April 2011
[153] "Dependency analysis," in *http://publib.boulder.ibm.com/infocenter/sr/v6r0/index.jsp?topic=/com.ibm.sr.doc/twsr_mansrvce_governanceuserguide05.html*, Last accessed:April 2011.
[154] "NDepend," in *http://www.ndepend.com/*, Last accessed: April 2011.
[155] "Understanding and Using Assemblies and Namespaces in .NET," in *http://msdn.microsoft.com/en-us/library/ms973231.aspx*, Last accessed: April 2011.
[156] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, 1976.
[157] "System Namespaces (.Net framework)," in *http://msdn.microsoft.com/en-us/library*, Last accessed: April 2011.
[158] "Building Testable ASP.NET MVC Applications," in *http://msdn.microsoft.com*, Last accessed: April 2011.
[159] "Microsoft Office XP primary interop assemblies (PIAs)," in *http://support.microsoft.com/kb/328912*, Last accessed: April 2011
[160] "mscorelib," in *http://dll.paretologic.com/detail.php/mscorlib*, Last accessed: Aprill 2011.
[161] M. Aditya, *Foundations of Software Testing*: Dorling Kindersley (India) Pvt. Ltd, 2008.
[162] S. Sanderson, *ASP.NET MVC framework preview*: Apress, 2010.
[163] P. Hamill, *Unit test frameworks*: O'Reilly Media Inc., 2005.
[164] "NUnit framework," in *http://www.nunit.org/*, Last accessed: April 2011.
[165] "Testing Google Earth " in *http://earth.google.com/getest.html*, Last accessed:April 2011
[166] "NCover," in *http://www.ncover.com/*, Last accessed: April 2011.

[167] "Test Impact Analysis," in *http://msdn.microsoft.com/en-us/library/dd286598.aspx*, Last accessed: April 2011.

[168] N. Koochakzadeh and V. Garousi, "TeCReVis: A Tool for Test Coverage and Test Redundancy Visualization," in *International Conference on Testing: Academic and Industrial Conference - Practice and Research Techniques*, 2010.

[169] A. M. Memon, "A Comprehensive Framework for Testing Graphical User Interfaces," PhD thesis, University of Pittsburgh, 2001.

[170] A. M. Memon, "GUI Testing: Pitfalls and Process," Computer, vol. 35, no. 8, pp. 87-88, 2002.

[171] A. M. Memon, "Advances in GUI Testing," Advances in Computers, vol. 58, no. 1, pp. 149-201, 2003.

[172] "Rational Functional Tester," in *http://www-01.ibm.com/software/awdtools/tester/functional/#*, Last accessed: April 2011.

[173] "Testing the User Interface with Automated UI Tests," in *http://msdn.microsoft.com/en-us/library/dd286726.aspx*, Last accessed: April 2011.

[174] "NUnitForms," in *http://nunitforms.sourceforge.net/*, Last Accessed: April 2011.

[175] "GUITAR--a GUI Testing frAmewoRk," in *http://guitar.sourceforge.net/*, Last accessed: April 2011.

[176] "Ranorex Studio," in *http://www.ranorex.com/products/ranorex-studio-test-automation-environment.html*, Last accessed: April 2011.

[177] "GUI testing," in *http://en.wikipedia.org/wiki/Graphical_user_interface_testing#cite_note-eight-8*, Last accessed: April 2011.

[178] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage Criteria for GUI Testing," in *Software Engineering conference held jointly with ACM SIGSOFT symposium on Foundations of software engineering*, 2001, pp. 256-267.

[179] "Behaviour-Driven Development," in *http://behaviour-driven.org*, Last accessed: April 2011.

Appendix A: Primary Studies and Their Type of Evidence

The table below shows the summary of the primary studies focus and the type of evidences they presented.

Type of evidence	Ref	Context/domain	Paper's main focus
Expert view without empirical backup	[3]	General	Identification of the gap between software engineers and scientists
	[5]	General	Identification of problematic issues in scientific computing
	[33]	General	Improving SS development
	[105]	Automotive	Developing software for automotive industry
	[74]	General	Practices for computational scientists
	[109]	General	Problem solving environments
	[101]	high performance computing	Identifying different risks
	[100]	high performance computing	SE for high performance computing
	[113]	high	
	[112]	high performance computing	
	[103]	Biology	Suggesting best practices for SS development
	[7]	Physical system modeling and simulation	Issues of developing SS
Case study	[88]	general	Practices to improve CSE software
Field study	[8]	Collaboration among software engineers and financial mathematicians, earth and planetary scientists, space scientists and molecular biologist	Differences between scientists and software engineers
	[80]	software engineers developing software for space scientists and biologists	Challenges of software engineers
	[81]	Biology	Culture and cooperation problems in SS

			development
	[20]	Collaboration among software engineers and financial mathematicians, earth and planetary scientists, space scientists and molecular biologist	Issues faced by SS developers
Case study	[23]	high performance computing	Identification of the steps and tools in developing high-performance software
	[68]	Mathematics	Proposing a methodology based on software requirement specification
	[89]	Imaging software development	Usability and user-centered design
	[24]	large-scale parallel code development running on high end computing systems	Developing large scale parallel software
	[102]	Requirement specification for beam analysis software development	Investigating the fact that the reliability of engineering computation can be significantly improved by adopting software engineering methodologies for requirements analysis and specification
	[66]	Scientific imaging software	Investigating the complexity of design
	[79]	Space scientific software development	Investigating the case where software engineers developing software for research scientists, using a traditional, staged, document-led methodology
	[97]	Multiphysics simulation	Modeling the input space for testing
	[29]	Developing weather forcasting code	Investigate the code development challenges and the tools used
Experience report	[94]	Parallel computation over data sets	Proposing and characterizing workflow systems
	[131]	Large-scale physics application	Using python in SS development
	[95]	developing control and data acquisition software for the SRS's (Synchrotron Radiation Source) experimental stations	SS development experiences and practices
	[96]	Developing scientific workflow	Automating scientific workflow

		management software system for collecting, analyzing, and managing data produced by sensors and other instruments and for creating subsequent reports	
	[104]	Biomedical software development	Investigating adopting agile method
	[107]	Bioinformatics	Introduction and adoption of agile method
	[87]	General	Proposing different integration strategies for computational science and engineering software
	[129]	Large scale multi-physics computational simulations	Lessons learned from developing large scale multi-physics computational simulations
	[108]	evaluating the performance of a numerical scheme to solve a model advection-diffusion problem	Adopting XP practices
	[31]	Computational biology and bioinformatics	Details of developing software for computational biology and bioinformatics
Case study and survey	[34]	High-Performance-Computing	Characterizing high-performance computing community
Survey	[26]	General	Developing scientific and computing software
	[77]	A mixture of engineering and scientific disciplines	Identifying different types of risks in testing SS development
	[36]	Computational chemistry	Surveying on how and where to integrate SE with computational science
	[86]	General	Characterization of SS
	[78]	A mixture of engineering and scientific disciplines	Dealing with risk
	[50]	General	Surveying how scientists develop and use SS
Exploratory study, interviews	[35]	Bioinformatics	Investigating the differences of software development by biologists and computer scientists
Illustration of ideas	[83]	General	Automated software testing for Matlab
Experiment	[128]	seismic data processing	Investigating errors in SS

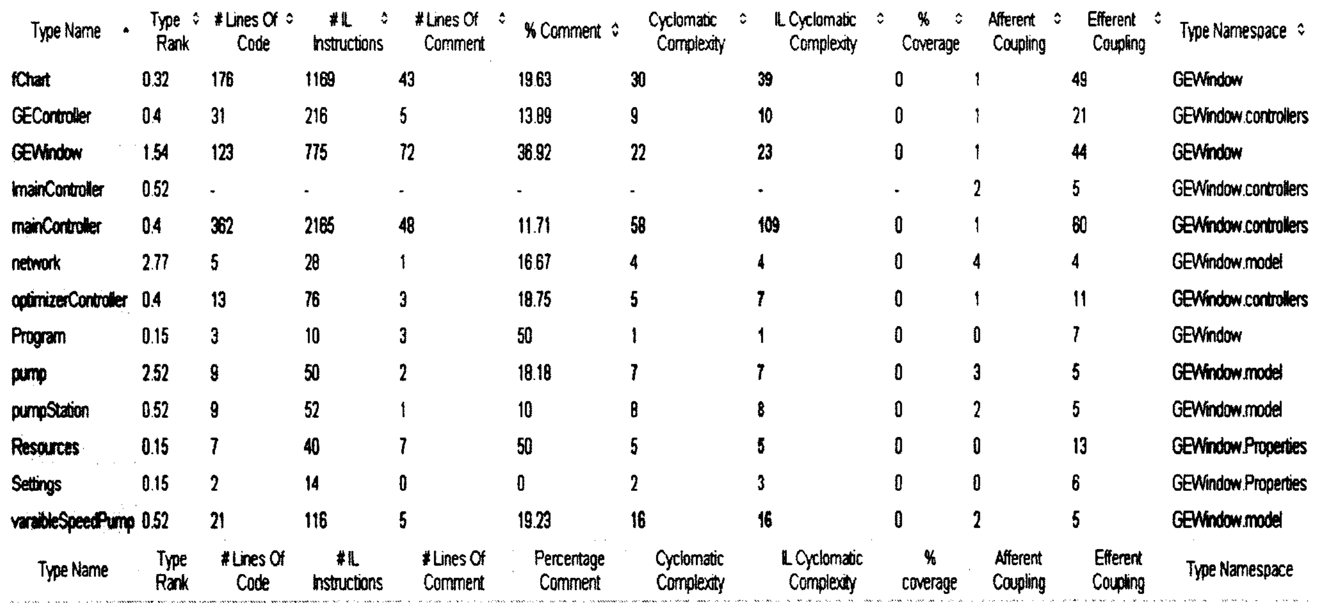
Systematic mapping	[49]	Embedded Software and Mobile Robot Software Development	Software engineering for embedded systems
Experiences and interviews	[117]	Developing a framework which can be used for assessment of how future alternative agricultural and environmental polices affect sustainable development in Europe	Investigation of the risks in a modeling framework and how to address them
Comparison	[72]	General	Comparing C++ and Fortran
	[71]	General	Comparing C++ and Fortran features
Interview and experience	[70]	Different scientific disciplines	Investigating quality assessment practices
Concept implementation and case study	[93]	Image retrieval-poison solver	Generative programming for SS developments
Review	[51]	General	CSE best practices
Concept implementation	[63]	Flight software development	Presenting new techniques for making requirements specifications precise, concise, unambiguous, and easy to check for completeness and consistency
	[98]	Quantom chemistry application development	Component-based architecture in quantum chemistry SC
	[111]	Multi-physics simulation	Proposing a framework for multi-physics simulations
	[116]	Dynamic-systems simulation	Introduction on using patterns for SS
	[119]	Computational life sciences	Presenting design patterns for SS and explaining their benefits
	[125]	High-performance scientific computing	Investigating the incorporation of message passing systems into component-based systems
	[25]	General	Presenting a methodology for development of the requirements for general purpose scientific computing software
	[91]	General	Proposing a framework to involve the domain experts in design
	[118]	Plasma physics	The use of design patterns
	[92]	Dealing with legacy scientific code	Integrating architectural constraints with legacy SS
	[122]	Scientifc library implementation	Proposing the use of a compiler to automatically optimize software library implementations

	[123]	General	Proposing a new architecture for SC application development
	[124]	High-performance scientific computing	Proposing a standard to support interoperability among high-performance scientific components
	[67]	Builidng scientifc software models	Integrating scientific applications
	[126]	High-performance scientific simulation	Developing SS component technology
	[127]	High end scientific computing	Presenting the Common Component Architecture for managing the complexity in high-performance scientific computing
	[110]	Generating scietific code	Using computer algebra systems to automatically generate a computer program
	[99]	High-performance scientific computing	Introducing a tool to perform rapid component prototyping while maintaining robust software engineering practices
	[106]	Langauge interoperability	Proposing an approach to fill the language gap in SS
	[130]	Large-scale parallel molecular dynamics simulations	Automatic SS scripting
	[75]	General	Testing SS
	[76]	General	Proposing mutation sensitivity testing
	[84]	General	Proposing the use of code mutation for testing SS
	[120]	Biology	Integrating a technological and design approach to support SS evolution
	[114]	General	Managing individualist programmer
	[115]	General	Proposing a new template for requirement specification
	[121]	Satellite data processing software	Using OO technology for the design of satellite data processing software
	[85]	Image segmentation software	Automated verification and validation technique for image segmentation
	[90]	Scientific workflow management system development	Proposing a scientific workflow management system

Table 30: Primary studies main focus and their type of evidence

Appendix B: Dependency Analysis

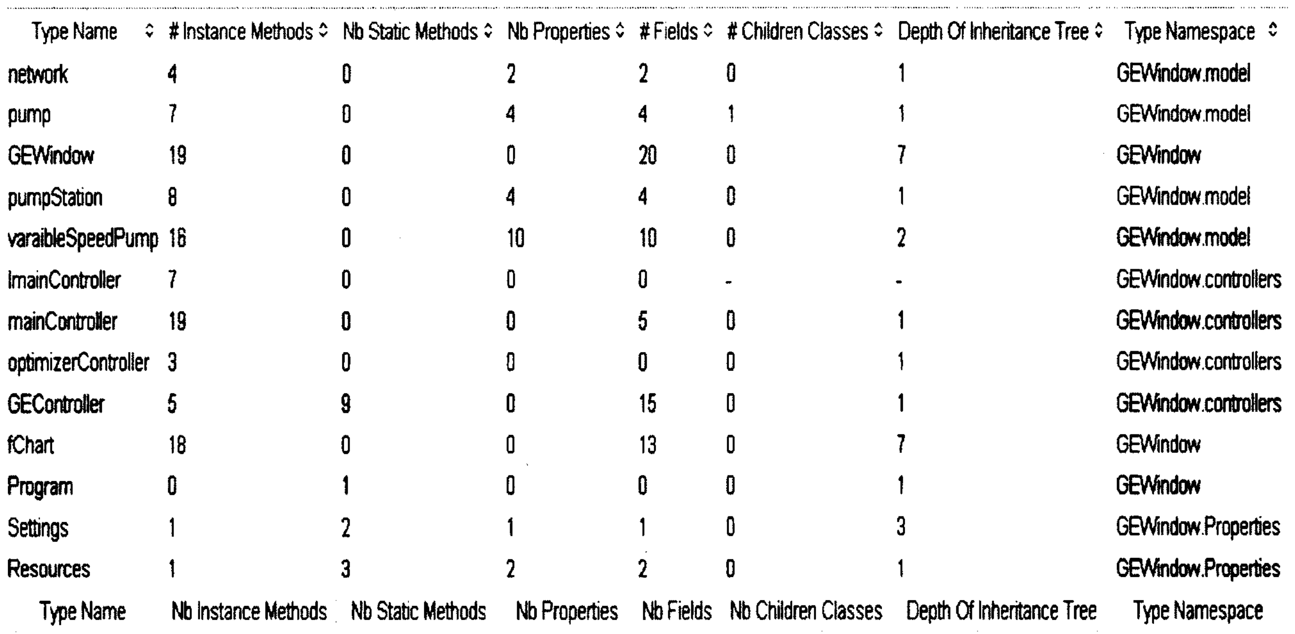
Type Metrics: Code Quality



Type Name ▴	Type Rank ◇	# Lines Of Code ◇	# IL Instructions ◇	# Lines Of Comment ◇	% Comment ◇	Cyclomatic Complexity ◇	IL Cyclomatic Complexity ◇	% Coverage ◇	Afferent Coupling ◇	Efferent Coupling ◇	Type Namespace ◇
fChart	0.32	176	1169	43	19.63	30	39	0	1	49	GEWindow
GEController	0.4	31	216	5	13.89	9	10	0	1	21	GEWindow.controllers
GEWindow	1.54	123	775	72	36.92	22	23	0	1	44	GEWindow
lmainController	0.52	-	-	-	-	-	-	-	2	5	GEWindow.controllers
mainController	0.4	362	2165	48	11.71	58	109	0	1	60	GEWindow.controllers
network	2.77	5	28	1	16.67	4	4	0	4	4	GEWindow.model
optimizerController	0.4	13	76	3	18.75	5	7	0	1	11	GEWindow.controllers
Program	0.15	3	10	3	50	1	1	0	0	7	GEWindow
pump	2.52	9	50	2	18.18	7	7	0	3	5	GEWindow.model
pumpStation	0.52	9	52	1	10	8	8	0	2	5	GEWindow.model
Resources	0.15	7	40	7	50	5	5	0	0	13	GEWindow.Properties
Settings	0.15	2	14	0	0	2	3	0	0	6	GEWindow.Properties
varaibleSpeedPump	0.52	21	116	5	19.23	16	16	0	2	5	GEWindow.model
Type Name	Type Rank	# Lines Of Code	# IL Instructions	# Lines Of Comment	Percentage Comment	Cyclomatic Complexity	IL Cyclomatic Complexity	% coverage	Afferent Coupling	Efferent Coupling	Type Namespace

Figure 54: Code Quality measures

Types Metrics: Code Members and Inheritance



Type Name ≎	# Instance Methods ≎	Nb Static Methods ≎	Nb Properties ≎	# Fields ≎	# Children Classes ≎	Depth Of Inheritance Tree ≎	Type Namespace ≎
network	4	0	2	2	0	1	GEWindow.model
pump	7	0	4	4	1	1	GEWindow.model
GEWindow	19	0	0	20	0	7	GEWindow
pumpStation	8	0	4	4	0	1	GEWindow.model
varaibleSpeedPump	16	0	10	10	0	2	GEWindow.model
ImainController	7	0	0	0	-	-	GEWindow.controllers
mainController	19	0	0	5	0	1	GEWindow.controllers
optimizerController	3	0	0	0	0	1	GEWindow.controllers
GEController	5	9	0	15	0	1	GEWindow.controllers
fChart	18	0	0	13	0	7	GEWindow
Program	0	1	0	0	0	1	GEWindow
Settings	1	2	1	1	0	3	GEWindow.Properties
Resources	1	3	2	2	0	1	GEWindow.Properties
Type Name	Nb Instance Methods	Nb Static Methods	Nb Properties	Nb Fields	Nb Children Classes	Depth Of Inheritance Tree	Type Namespace

Figure 55: Code Members and Inheritance

Treemap metric view

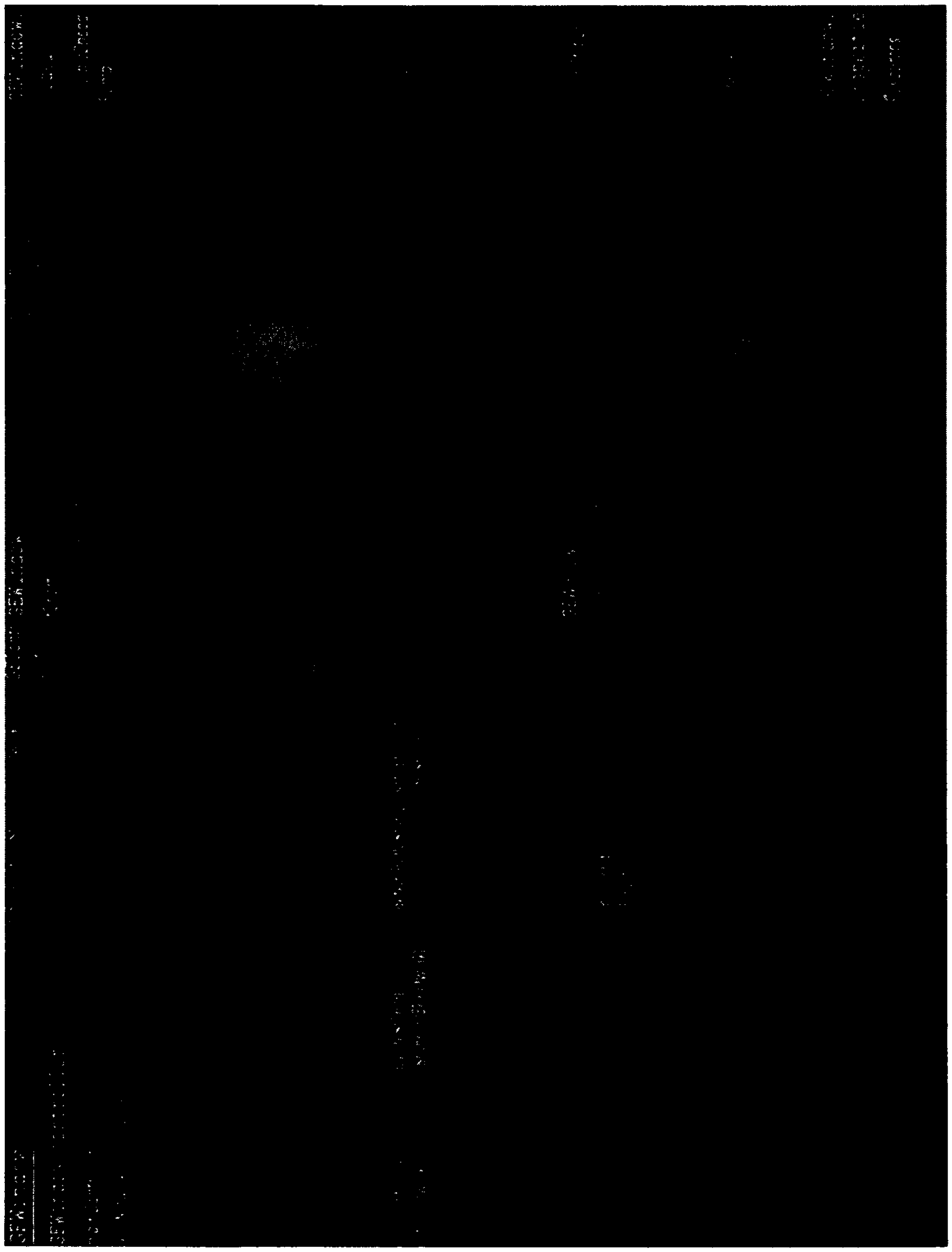


Figure 56: Treemap metric view

Dependency Graphs

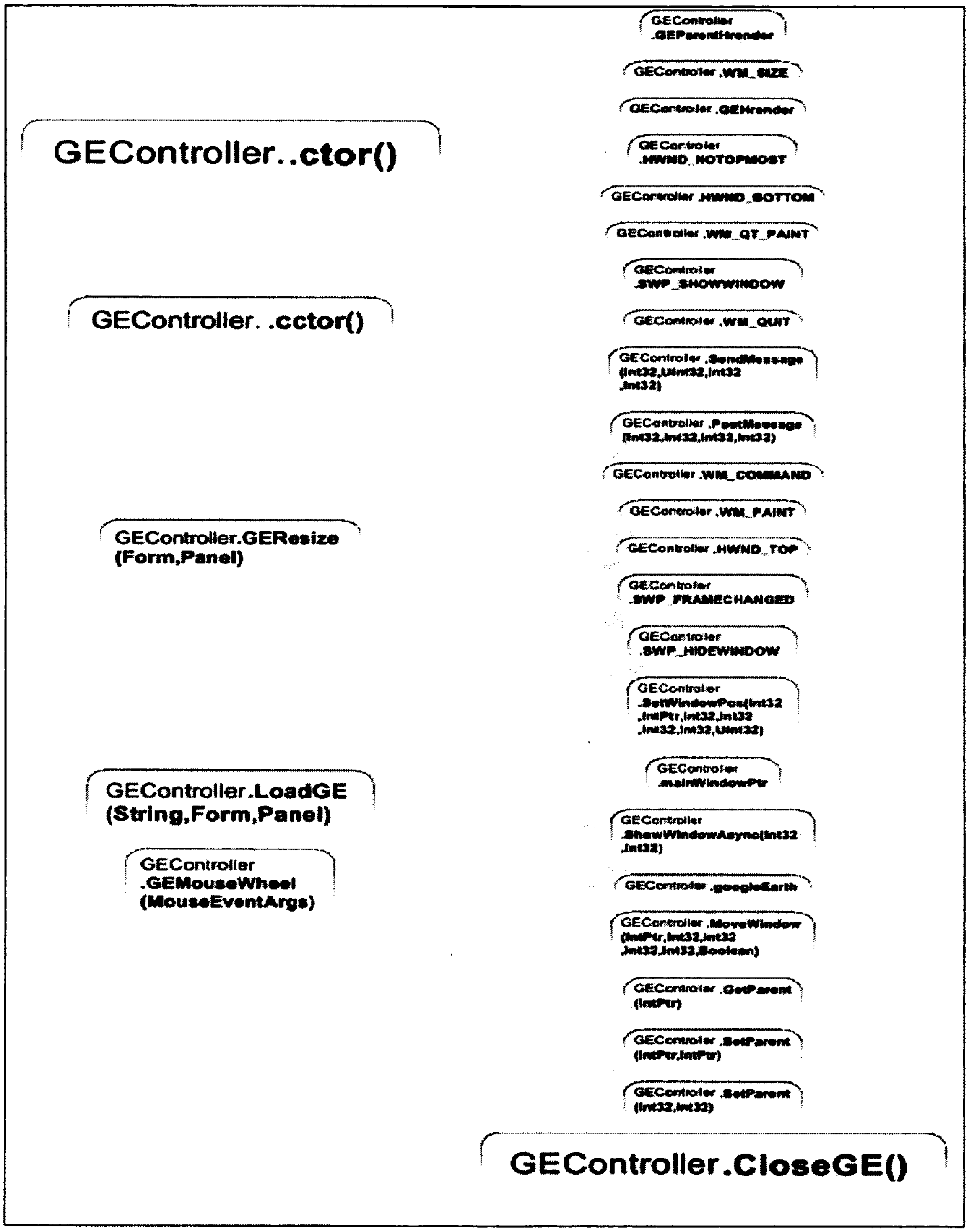

GEController..ctor()
GEController. .cctor()
GEController.GEResize (Form,Panel)
GEController.LoadGE (String,Form,Panel)
GEController .GEMouseWheel (MouseEventArgs)
GEController .GEParentHrender
GEController .WM_SIZE
GEController .GEHrender
GEController .HWND_NOTOPMOST
GEController .HWND_BOTTOM
GEController .WM_QT_PAINT
GEController .SWP_SHOWWINDOW
GEController .WM_QUIT
GEController .SendMessage (Int32,UInt32,Int32 ,Int32)
GEController .PostMessage (Int32,Int32,Int32,Int32)
GEController .WM_COMMAND
GEController .WM_PAINT
GEController .HWND_TOP
GEController .SWP_FRAMECHANGED
GEController .SWP_HIDEWINDOW
GEController .SetWindowPos(Int32 ,IntPtr,Int32,Int32 ,Int32,Int32,UInt32)
GEController .mainWindowPtr
GEController .ShowWindowAsync(Int32 ,Int32)
GEController .googleEarth
GEController .MoveWindow (IntPtr,Int32,Int32 ,Int32,Int32,Boolean)
GEController .GetParent (IntPtr)
GEController .SetParent (IntPtr,IntPtr)
GEController .SetParent (Int32,Int32)
GEController.CloseGE()

Figure 57: Dependency graph, within GEController

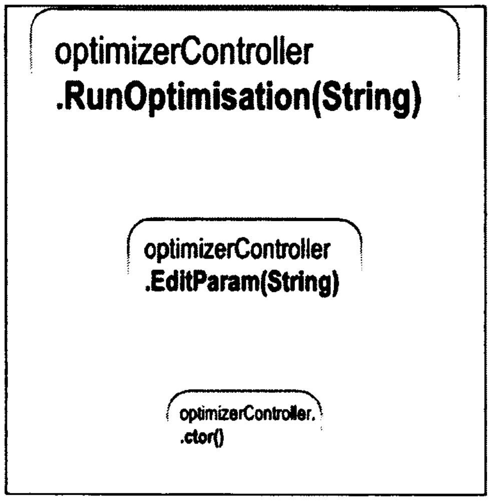


Figure 58: Dependency graph, within optimizerController

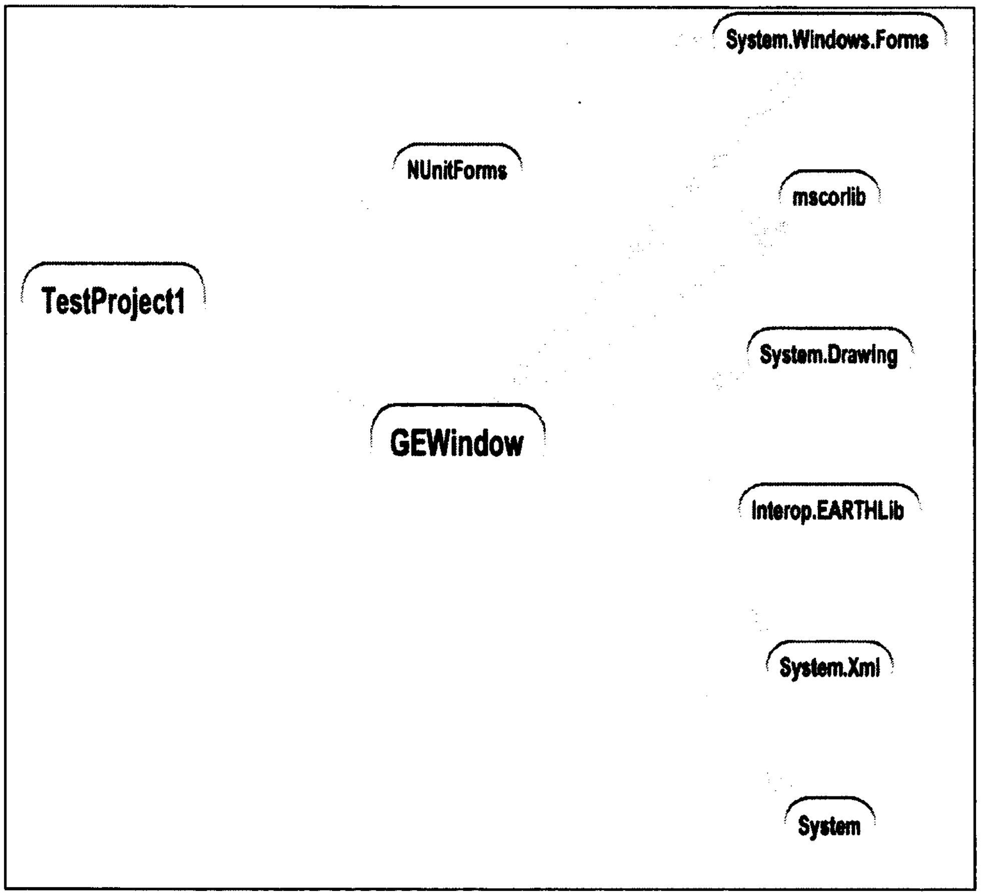


Figure 59: Dependency graph among the application's assemblies and the test assembly

Dependency Matrix

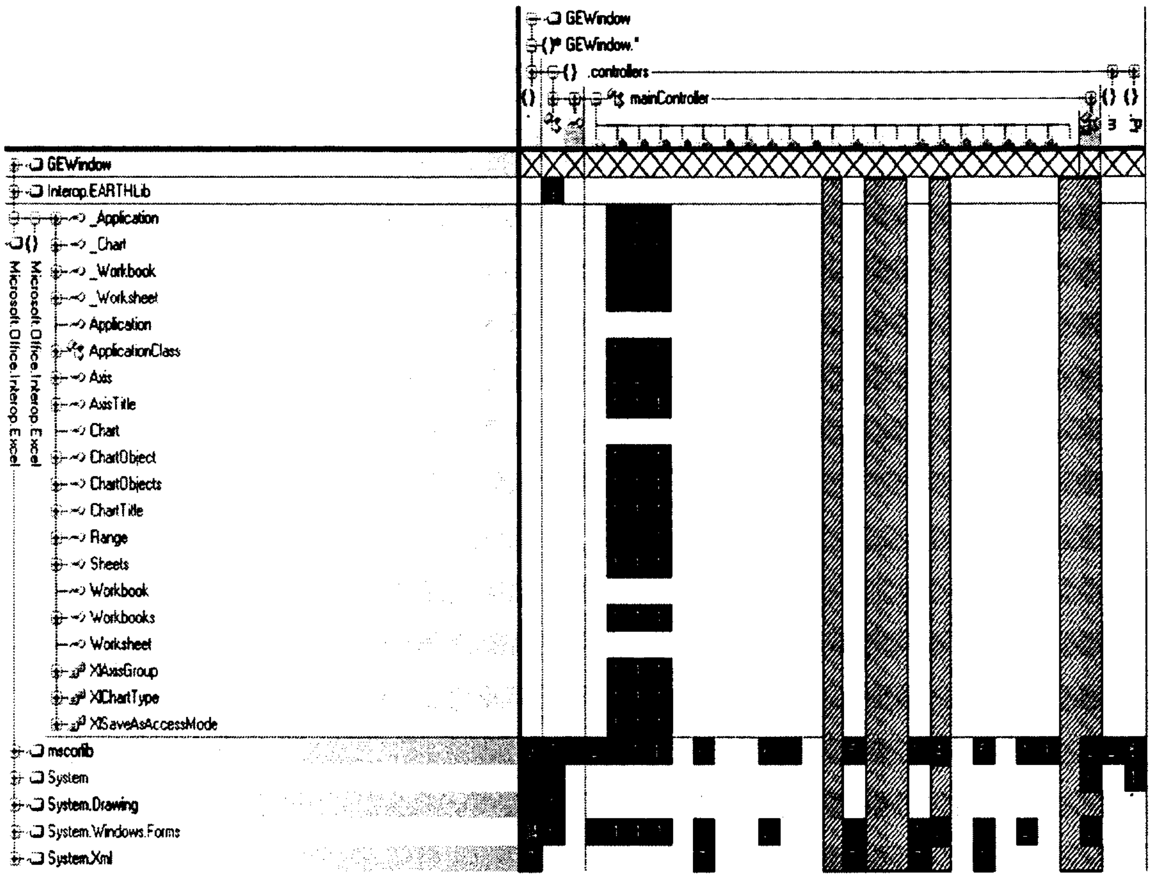


Figure 60: Dependency matrix

Code Visualisation

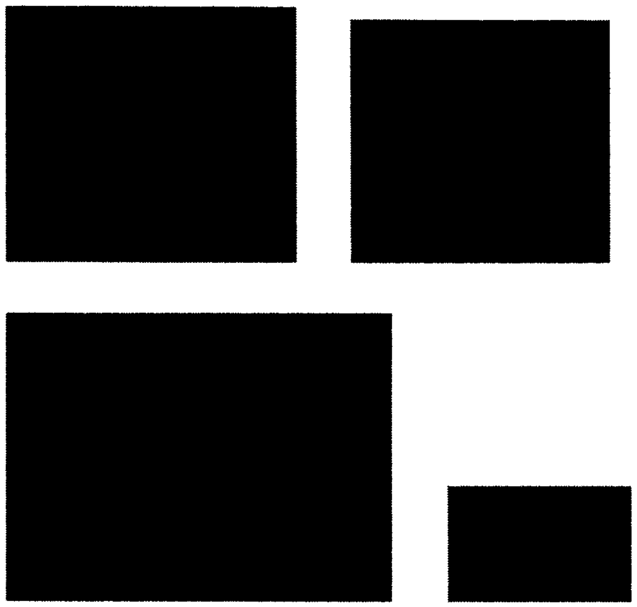


Figure 61: Code city "Top-down" perspective, top left: Controllers, top right: Models, bottom left: Views, bottom right: properties

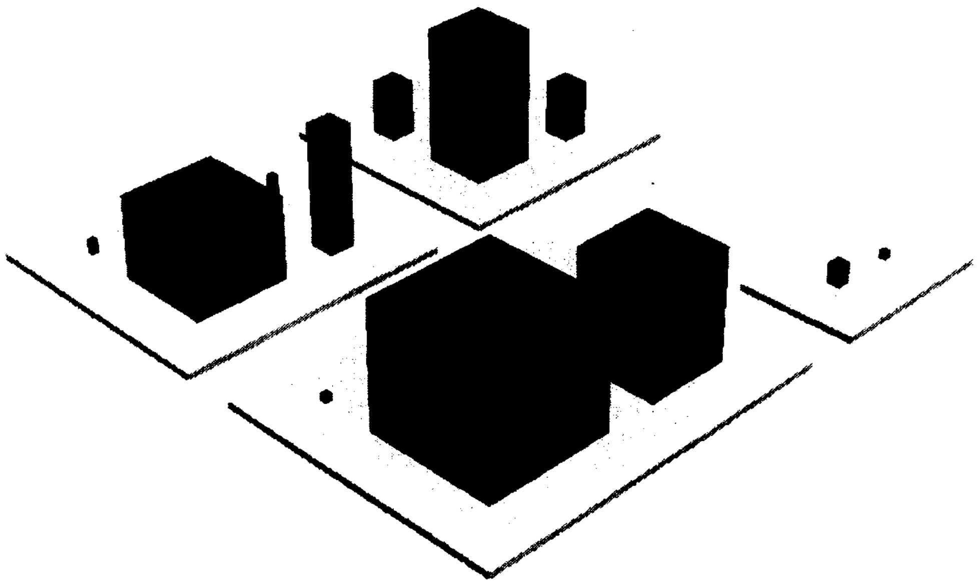


Figure 62: Code city "Isometric" perspective

Appendix C: Test Cases

createOptChart method test cases

```
using GEWindow.controllers;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using GEWindow.model;
using Excel = Microsoft.Office.Interop.Excel;

namespace TestProject1
{
    /// <summary>
    ///This is a test class for mainControllerTest and is intended
    ///to contain all mainControllerTest Unit Tests
    ///</summary>
    [TestClass()]
    public class createOptChartTest
    {
        private TestContext testContextInstance;

        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current test run.
        ///</summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        /// <summary>
        ///A test for createStChart
        ///</summary>
        [TestMethod()]
        public void createOptChartTestImagePath()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 10; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            string expected = @"h:\Program Files\Microsoft Visual Studio
10.0\Common7\IDE" + "\\img10.bmp"; // TODO: Initialize to an appropriate value
            string actual;
            actual = target.createOptChart(iCounter, nt);
            Assert.AreEqual(expected, actual);
            //Assert.Inconclusive("Verify the correctness of this test method.");
        }
        [TestMethod()]
        public void createOptChartTestNominalHeader1()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter,nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);
            //xlWorkBook = xlApp.Workbooks.Open(@"C:\Program Files\TestDriven.NET
3\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);
            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[1,
7]).Value2.ToString();
            string expected = "Total Cost";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestNominalOpti()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
            string actual = ((Excel.Range)xlWorkSheet.Cells[2,
4]).Value2.ToString();
            string expected = "0";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestNominalScada()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[3,
3]).Value2.ToString();
            string expected = "392";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestLowBoundaryOpti()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);
```

```
            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[2,
2]).Value2.ToString();
            string expected = "640";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestLowBoundaryScada()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[3,
2]).Value2.ToString();
            string expected = "415";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestHighBoundaryOpti()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[2,
7]).Value2.ToString();
            string expected = "1364.87293";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartTestHighBoundaryScada()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[3,
7]).Value2.ToString();
            string expected = "3033";
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void createOptChartNimnalHeader2()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            int iCounter = 1; // TODO: Initialize to an appropriate value
            network nt = new network(); // TODO: Initialize to an appropriate
value
            target.readNetworkData(nt);
            target.createOptChart(iCounter, nt);

            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;

            xlApp = new Excel.ApplicationClass();
            xlWorkBook = xlApp.Workbooks.Open(@"h:\Program Files\Microsoft Visual
Studio 10.0\Common7\IDE\" + "imageExcel.xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false, false, 0, true,
1, 0);

            xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);

            string actual = ((Excel.Range)xlWorkSheet.Cells[2,
1]).Value2.ToString();
            string expected = "Optimization";
            Assert.AreEqual(expected, actual);
        }
    }
}
```

speedRoots method test cases

```
using GEWindow.controllers;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
//using NUnit.Framework;

namespace TestProject1
{

    /// <summary>
    ///This is a test class for mainControllerTest and is intended
    ///to contain all mainControllerTest Unit Tests
    ///</summary>
    [TestClass()]
    public class speedRootsTest
    {
        private TestContext testContextInstance;
        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current test run.
        ///</summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPPPPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
```

```
an appropriate value
            double a = 45.56; // TODO: Initialize to an appropriate value
            double b = 234; // TODO: Initialize to an appropriate value
            double c = 1; // TODO: Initialize to an appropriate value
            double cons = 23.7; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 450; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestPPPNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 45.6; // TODO: Initialize to an appropriate value
            double b = 76.56; // TODO: Initialize to an appropriate value
            double c = 5679.9; // TODO: Initialize to an appropriate value
            double cons = -19.9; // TODO: Initialize to an appropriate value
            double H = 467; // TODO: Initialize to an appropriate value
            double Q = 4; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestPPNPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 56; // TODO: Initialize to an appropriate value
            double b = 6789; // TODO: Initialize to an appropriate value
            double c = -78.9; // TODO: Initialize to an appropriate value
            double cons = 98; // TODO: Initialize to an appropriate value
            double H = 9; // TODO: Initialize to an appropriate value
            double Q = 100; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPPNNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 45; // TODO: Initialize to an appropriate value
            double b = 300000; // TODO: Initialize to an appropriate value
            double c = -2; // TODO: Initialize to an appropriate value
            double cons = -9.6; // TODO: Initialize to an appropriate value
            double H =50; // TODO: Initialize to an appropriate value
            double Q = 45.9; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPNPPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 45; // TODO: Initialize to an appropriate value
            double b = -89989; // TODO: Initialize to an appropriate value
            double c = 8.9; // TODO: Initialize to an appropriate value
            double cons = 78; // TODO: Initialize to an appropriate value
            double H = 45; // TODO: Initialize to an appropriate value
            double Q = 10; // TODO: Initialize to an appropriate value
            double expected = 101111.23; // TODO: Initialize to an appropriate
value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPNPNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 0.0124; // TODO: Initialize to an appropriate value
            double b = -0.4903; // TODO: Initialize to an appropriate value
            double c = 806.5363; // TODO: Initialize to an appropriate value
            double cons = -0.8713; // TODO: Initialize to an appropriate value
            double H = 78; // TODO: Initialize to an appropriate value
            double Q = 8; // TODO: Initialize to an appropriate value
            double expected = 0.31; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPNNPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 0.0124; // TODO: Initialize to an appropriate value
            double b = -0.4903; // TODO: Initialize to an appropriate value
            double c = -806.5363; // TODO: Initialize to an appropriate value
            double cons = 0.8713; // TODO: Initialize to an appropriate value
            double H = 500; // TODO: Initialize to an appropriate value
            double Q = 200; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestPNNNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = 0.0124; // TODO: Initialize to an appropriate value
            double b = -0.4903; // TODO: Initialize to an appropriate value
            double c = -806.5363; // TODO: Initialize to an appropriate value
            double cons = -0.8713; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 80; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestNPPPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0124; // TODO: Initialize to an appropriate value
            double b = 0.4903; // TODO: Initialize to an appropriate value
            double c = 806.5363; // TODO: Initialize to an appropriate value
            double cons = 0.8713; // TODO: Initialize to an appropriate value
            double H = 0.1; // TODO: Initialize to an appropriate value
            double Q = 100; // TODO: Initialize to an appropriate value
            double expected = 0.36; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }
        /// <summary>
        ///A test for speedRoots
        ///</summary>
        [TestMethod()]
        public void speedRootsTestNPPNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = 0.4345; // TODO: Initialize to an appropriate value
            double c = 926.9063; // TODO: Initialize to an appropriate value
            double cons = -5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 10; // TODO: Initialize to an appropriate value
            double expected = 0.33; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNPNPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = 11110.4345; // TODO: Initialize to an appropriate value
            double c = -926.9063; // TODO: Initialize to an appropriate value
            double cons = 5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 10; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNPNNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = 0.4345; // TODO: Initialize to an appropriate value
            double c = -926.9063; // TODO: Initialize to an appropriate value
            double cons = -5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 80; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNNPPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = -0.4345; // TODO: Initialize to an appropriate value
            double c = 926.9063; // TODO: Initialize to an appropriate value
            double cons = 5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 10; // TODO: Initialize to an appropriate value
            double expected = 0.32; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNNPNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = -0.4345; // TODO: Initialize to an appropriate value
            double c = 926.9063; // TODO: Initialize to an appropriate value
            double cons = -5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 80; // TODO: Initialize to an appropriate value
            double expected = 0.38; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNNNPPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = -0.4345; // TODO: Initialize to an appropriate value
            double c = -926.9063; // TODO: Initialize to an appropriate value
            double cons = 5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 90; // TODO: Initialize to an appropriate value
            double expected =0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }

        [TestMethod()]
        public void speedRootsTestNNNNPP()
        {
            mainController target = new mainController(); // TODO: Initialize to
an appropriate value
            double a = -0.0022; // TODO: Initialize to an appropriate value
            double b = -0.4345; // TODO: Initialize to an appropriate value
            double c = -926.9063; // TODO: Initialize to an appropriate value
            double cons = -5.1234; // TODO: Initialize to an appropriate value
            double H = 100; // TODO: Initialize to an appropriate value
            double Q = 80; // TODO: Initialize to an appropriate value
            double expected = 0; // TODO: Initialize to an appropriate value
            double actual;
            actual = target.speedRoots(a, b, c, cons, H, Q);
            Assert.AreEqual(expected, actual);
        }
    }
}
```